

# DataLab Cup 3: Reverse Image Caption

Shan-Hung Wu & DataLab  
Fall 2023

## Text to Image

Platform: [Kaggle](#)

### Overview

In this work, we are interested in translating text in the form of single-sentence human-written descriptions directly into image pixels. For example, "**this flower has petals that are yellow and has a ruffled stamen**" and "**this pink and yellow flower has a beautiful yellow center with many stamens**". You have to develop a novel deep architecture and GAN formulation to effectively translate visual concepts from characters to pixels.

More specifically, given a set of texts, your task is to generate reasonable images with size 64x64x3 to illustrate the corresponding texts. Here we use [Oxford-102 flower dataset](#) and its [paired texts](#) as our training dataset.



this flower has petals that are yellow and has a ruffled stamen



this pink and yellow flower has a beautiful yellow center with many stamens

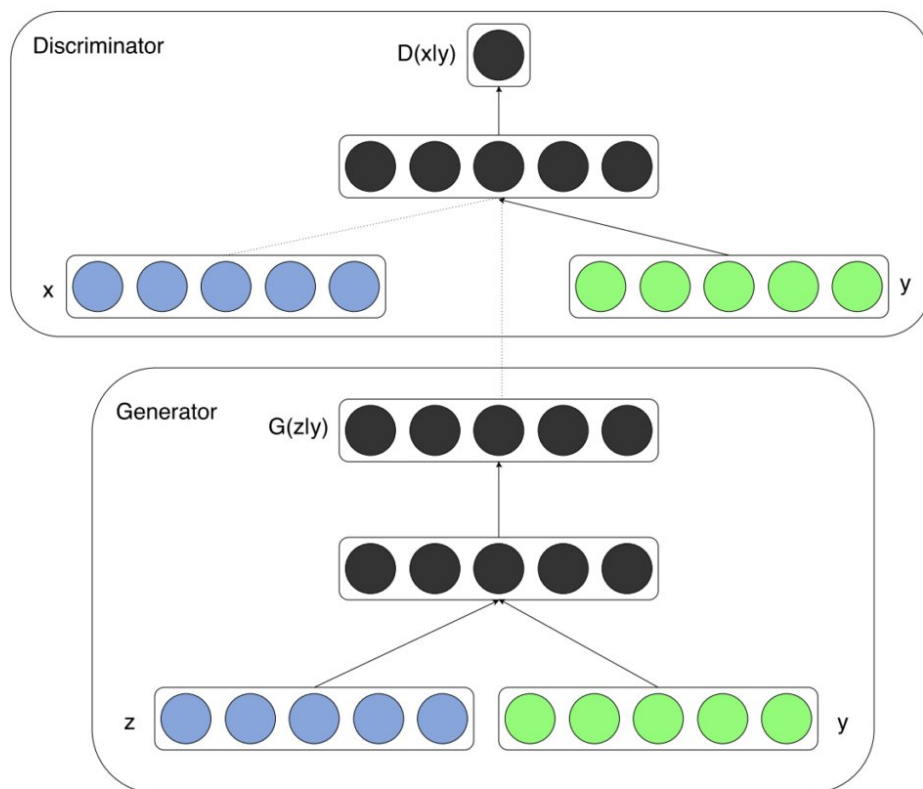
- 7370 images as training set, where each images is annotated with at most 10 texts.
- 819 texts for testing. You must generate 1 64x64x3 image for each text.

### Conditional GAN

Given a text, in order to generate the image which can illustrate it, our model must meet several requirements:

1. Our model should have ability to understand and extract the meaning of given texts.
  - Use RNN or other language model, such as BERT, ELMo or XLNet, to capture the meaning of text.
2. Our model should be able to generate image.
  - Use GAN to generate high quality image.
3. GAN-generated image should illustrate the text.
  - Use conditional-GAN to generate image conditioned on given text.

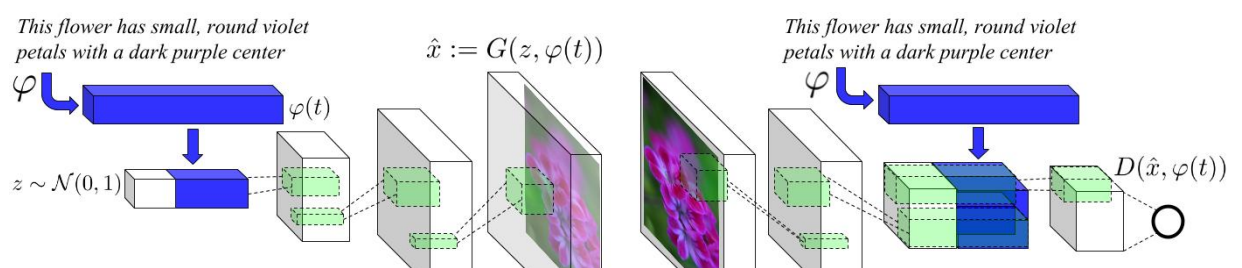
Generative adversarial nets can be extended to a conditional model if both the generator and discriminator are conditioned on some extra information  $y$ . We can perform the conditioning by feeding  $y$  into both the discriminator and generator as additional input layer.



There are two motivations for using some extra information in a GAN model:

1. Improve GAN.
2. Generate targeted image.

Additional information that is correlated with the input images, such as class labels, can be used to improve the GAN. This improvement may come in the form of more stable training, faster training, and/or generated images that have better quality.



```
In [ ]: from __future__ import absolute_import, division, print_function, unicode_literals
import tensorflow as tf
from tensorflow.keras import layers
import os
os.environ['TF_CPP_MIN_LOG_LEVEL'] = '2'
import string
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import PIL
import random
import time
from pathlib import Path

import re
from IPython import display
```

```
In [ ]: gpus = tf.config.experimental.list_physical_devices('GPU')
if gpus:
    try:
        # Restrict TensorFlow to only use the first GPU
        tf.config.experimental.set_visible_devices(gpus[0], 'GPU')

        # Currently, memory growth needs to be the same across GPUs
        for gpu in gpus:
            tf.config.experimental.set_memory_growth(gpu, True)
        logical_gpus = tf.config.experimental.list_logical_devices('GPU')
        print(len(gpus), "Physical GPUs,", len(logical_gpus), "Logical GPUs")
    except RuntimeError as e:
        # Memory growth must be set before GPUs have been initialized
        print(e)
```

## Preprocess Text

Since dealing with raw string is inefficient, we have done some data preprocessing for you:

- Delete text over `MAX_SEQ_LENGTH (20)` .
- Delete all punctuation in the texts.
- Encode each vocabulary in `dictionary/vocab.npy` .
- Represent texts by a sequence of integer IDs.
- Replace rare words by `<RARE>` token to reduce vocabulary size for more efficient training.
- Add padding as `<PAD>` to each text to make sure all of them have equal length to `MAX_SEQ_LENGTH (20)` .

It is worth knowing that there is no necessary to append `<ST>` and `<ED>` to each text because we don't need to generate any sequence in this task.

To make sure correctness of encoding of the original text, we can decode sequence vocabulary IDs by looking up the vocabulary dictionary:

- `dictionary/word2Id.npy` is a numpy array mapping word to id.
- `dictionary/id2Word.npy` is a numpy array mapping id back to word.

```
In [3]: dictionary_path = './dictionary'
vocab = np.load(dictionary_path + '/vocab.npy')
print('there are {} vocabularies in total'.format(len(vocab)))

word2Id_dict = dict(np.load(dictionary_path + '/word2Id.npy'))
id2word_dict = dict(np.load(dictionary_path + '/id2Word.npy'))
print('Word to id mapping, for example: %s -> %s' % ('flower', word2Id_dict['flower']))
print('Id to word mapping, for example: %s -> %s' % ('1', id2word_dict['1']))
print('Tokens: <PAD>: %s; <RARE>: %s' % (word2Id_dict['<PAD>'], word2Id_dict['<RARE>']))

there are 5427 vocabularies in total
Word to id mapping, for example: flower -> 1
Id to word mapping, for example: 1 -> flower
Tokens: <PAD>: 5427; <RARE>: 5428
```

```
In [4]: def sent2IdList(line, MAX_SEQ_LENGTH=20):
    MAX_SEQ_LIMIT = MAX_SEQ_LENGTH
    padding = 0

    # data preprocessing, remove all punctuation in the texts
    prep_line = re.sub('[%s]' % re.escape(string.punctuation), ' ', line.rstrip())
    prep_line = prep_line.replace('-', ' ')
    prep_line = prep_line.replace('-', ' ')
    prep_line = prep_line.replace(' ', ' ')
    prep_line = prep_line.replace('.', '')
    tokens = prep_line.split(' ')
    tokens = [
        tokens[i] for i in range(len(tokens))
        if tokens[i] != ' ' and tokens[i] != ''
    ]
    l = len(tokens)
    padding = MAX_SEQ_LIMIT - l

    # make sure length of each text is equal to MAX_SEQ_LENGTH, and replace the less com
    for i in range(padding):
        tokens.append('<PAD>')
    line = [
        word2Id_dict[tokens[k]]
        if tokens[k] in word2Id_dict else word2Id_dict['<RARE>']
        for k in range(len(tokens))
    ]

    return line

text = "the flower shown has yellow anther red pistil and bright red petals."
print(text)
print(sent2IdList(text))
```

```
the flower shown has yellow anther red pistil and bright red petals.
['9', '1', '82', '5', '11', '70', '20', '31', '3', '29', '20', '2', '5427', '5427', '5427', '5427', '5427', '5427', '5427', '5427', '5427', '5427']
```

## Dataset

For training, the following files are in dataset folder:

- `./dataset/text2ImgData.pkl` is a pandas dataframe with attribute 'Captions' and 'ImagePath'.
  - 'Captions': A list of text id list contain 1 to 10 captions.
  - 'ImagePath': Image path that store paired image.

- `./102flowers/` is the directory containing all training images.
- `./dataset/testData.pkl` is a pandas dataframe with attribute 'ID' and 'Captions', which contains testing data.

```
In [5]: data_path = './dataset'
df = pd.read_pickle(data_path + '/text2ImgData.pkl')
num_training_sample = len(df)
n_images_train = num_training_sample
print('There are %d image in training data' % (n_images_train))
```

There are 7370 image in training data

```
In [6]: df.head(5)
```

```
Out[6]:
```

	Captions	ImagePath
ID		
6734	[[9, 2, 17, 9, 1, 6, 14, 13, 18, 3, 41, 8, 11,...	./102flowers/image_06734.jpg
6736	[[4, 1, 5, 12, 2, 3, 11, 31, 28, 68, 106, 132,...	./102flowers/image_06736.jpg
6737	[[9, 2, 27, 4, 1, 6, 14, 7, 12, 19, 5427, 5427...	./102flowers/image_06737.jpg
6738	[[9, 1, 5, 8, 54, 16, 38, 7, 12, 116, 325, 3, ...	./102flowers/image_06738.jpg
6739	[[4, 12, 1, 5, 29, 11, 19, 7, 26, 70, 5427, 54...	./102flowers/image_06739.jpg

## Create Dataset by Dataset API

```
In [7]: # in this competition, you have to generate image in size 64x64x3
IMAGE_HEIGHT = 64
IMAGE_WIDTH = 64
IMAGE_CHANNEL = 3

def training_data_generator(caption, image_path):
    # Load in the image according to image path
    img = tf.io.read_file(image_path)
    img = tf.image.decode_image(img, channels=3)
    img = tf.image.convert_image_dtype(img, tf.float32)
    img.set_shape([None, None, 3])
    img = tf.image.resize(img, size=[IMAGE_HEIGHT, IMAGE_WIDTH])
    img.set_shape([IMAGE_HEIGHT, IMAGE_WIDTH, IMAGE_CHANNEL])
    caption = tf.cast(caption, tf.int32)

    return img, caption

def dataset_generator(filenamees, batch_size, data_generator):
    # Load the training data into two NumPy arrays
    df = pd.read_pickle(filenamees)
    captions = df['Captions'].values
    caption = []
    # each image has 1 to 10 corresponding captions
    # we choose one of them randomly for training
    for i in range(len(captions)):
        caption.append(random.choice(captions[i]))
    caption = np.asarray(caption)
    caption = caption.astype(np.int)
    image_path = df['ImagePath'].values
```

```

# assume that each row of `features` corresponds to the same row as `labels`.
assert caption.shape[0] == image_path.shape[0]

dataset = tf.data.Dataset.from_tensor_slices((caption, image_path))
dataset = dataset.map(data_generator, num_parallel_calls=tf.data.experimental.AUTOTUNE)
dataset = dataset.shuffle(len(caption)).batch(batch_size, drop_remainder=True)
dataset = dataset.prefetch(buffer_size=tf.data.experimental.AUTOTUNE)

return dataset

```

```

In [8]: BATCH_SIZE = 64
dataset = dataset_generator(data_path + '/text2ImgData.pkl', BATCH_SIZE, training_data_g

/tmp/ipykernel_29626/3313815207.py:28: DeprecationWarning: `np.int` is a deprecated alias
s for the builtin `int`. To silence this warning, use `int` by itself. Doing this will n
ot modify any behavior and is safe. When replacing `np.int`, you may wish to use e.g. `n
p.int64` or `np.int32` to specify the precision. If you wish to review your current use,
check the release note link for additional information.
Deprecated in NumPy 1.20; for more details and guidance: https://numpy.org/devdocs/relea
se/1.20.0-notes.html#deprecations
caption = caption.astype(np.int)

```

## Conditional GAN Model

As mentioned above, there are three models in this task, text encoder, generator and discriminator.

### Text Encoder

A RNN encoder that captures the meaning of input text.

- Input: text, which is a list of ids.
- Output: embedding, or hidden representation of input text.

```

In [9]: class TextEncoder(tf.keras.Model):
        """
        Encode text (a caption) into hidden representation
        input: text, which is a list of ids
        output: embedding, or hidden representation of input text in dimension of RNN_HIDDEN
        """
        def __init__(self, hparas):
            super(TextEncoder, self).__init__()
            self.hparas = hparas
            self.batch_size = self.hparas['BATCH_SIZE']

            # embedding with tensorflow API
            self.embedding = layers.Embedding(self.hparas['VOCAB_SIZE'], self.hparas['EMBED_
            # RNN, here we use GRU cell, another common RNN cell similar to LSTM
            self.gru = layers.GRU(self.hparas['RNN_HIDDEN_SIZE'],
                                   return_sequences=True,
                                   return_state=True,
                                   recurrent_initializer='glorot_uniform')

        def call(self, text, hidden):
            text = self.embedding(text)
            output, state = self.gru(text, initial_state = hidden)
            return output[:, -1, :], state

```



```
def initialize_hidden_state(self):
    return tf.zeros((self.hparas['BATCH_SIZE'], self.hparas['RNN_HIDDEN_SIZE']))
```

## Generator

A image generator which generates the target image illustrating the input text.

- Input: hidden representation of input text and random noise z with random seed.
- Output: target image, which is conditioned on the given text, in size 64x64x3.

```
In [10]: class Generator(tf.keras.Model):
        """
        Generate fake image based on given text(hidden representation) and noise z
        input: text and noise
        output: fake image with size 64*64*3
        """
        def __init__(self, hparas):
            super(Generator, self).__init__()
            self.hparas = hparas
            self.flatten = tf.keras.layers.Flatten()
            self.d1 = tf.keras.layers.Dense(self.hparas['DENSE_DIM'])
            self.d2 = tf.keras.layers.Dense(64*64*3)

        def call(self, text, noise_z):
            text = self.flatten(text)
            text = self.d1(text)
            text = tf.nn.leaky_relu(text)

            # concatenate input text and random noise
            text_concat = tf.concat([noise_z, text], axis=1)
            text_concat = self.d2(text_concat)

            logits = tf.reshape(text_concat, [-1, 64, 64, 3])
            output = tf.nn.tanh(logits)

            return logits, output
```

## Discriminator

A binary classifier which can discriminate the real and fake image:

1. Real image
  - Input: real image and the paired text
  - Output: a floating number representing the result, which is expected to be 1.
2. Fake Image
  - Input: generated image and paired text
  - Output: a floating number representing the result, which is expected to be 0.

```
In [11]: class Discriminator(tf.keras.Model):
        """
        Differentiate the real and fake image
        input: image and corresponding text
        output: labels, the real image should be 1, while the fake should be 0
        """
        def __init__(self, hparas):
```

```

super(Discriminator, self).__init__()
self.hparas = hparas
self.flatten = tf.keras.layers.Flatten()
self.d_text = tf.keras.layers.Dense(self.hparas['DENSE_DIM'])
self.d_img = tf.keras.layers.Dense(self.hparas['DENSE_DIM'])
self.d = tf.keras.layers.Dense(1)

def call(self, img, text):
    text = self.flatten(text)
    text = self.d_text(text)
    text = tf.nn.leaky_relu(text)

    img = self.flatten(img)
    img = self.d_img(img)
    img = tf.nn.leaky_relu(img)

    # concatenate image with paired text
    img_text = tf.concat([text, img], axis=1)

    logits = self.d(img_text)
    output = tf.nn.sigmoid(logits)

    return logits, output

```

```

In [12]: hparas = {
    'MAX_SEQ_LENGTH': 20,                # maximum sequence length
    'EMBED_DIM': 256,                    # word embedding dimension
    'VOCAB_SIZE': len(word2Id_dict),     # size of dictionary of captions
    'RNN_HIDDEN_SIZE': 128,              # number of RNN neurons
    'Z_DIM': 512,                        # random noise z dimension
    'DENSE_DIM': 128,                    # number of neurons in dense layer
    'IMAGE_SIZE': [64, 64, 3],           # render image size
    'BATCH_SIZE': 64,
    'LR': 1e-4,
    'LR_DECAY': 0.5,
    'BETA_1': 0.5,
    'N_EPOCH': 600,
    'N_SAMPLE': num_training_sample,     # size of training data
    'CHECKPOINTS_DIR': './checkpoints/demo', # checkpoint path
    'PRINT_FREQ': 1                      # printing frequency of loss
}

```

```

In [13]: text_encoder = TextEncoder(hparas)
generator = Generator(hparas)
discriminator = Discriminator(hparas)

```

## Loss Function and Optimization

Although the conditional GAN model is quite complex, the loss function used to optimize the network is relatively simple. Actually, it is simply a binary classification task, thus we use cross entropy as our loss.

```

In [14]: # This method returns a helper function to compute cross entropy loss
cross_entropy = tf.keras.losses.BinaryCrossentropy(from_logits=True)

```

```

In [15]: def discriminator_loss(real_logits, fake_logits):
    # output value of real image should be 1
    real_loss = cross_entropy(tf.ones_like(real_logits), real_logits)
    # output value of fake image should be 0

```



```

fake_loss = cross_entropy(tf.zeros_like(fake_logits), fake_logits)
total_loss = real_loss + fake_loss
return total_loss

```

```

def generator_loss(fake_output):
    # output value of fake image should be 0
    return cross_entropy(tf.ones_like(fake_output), fake_output)

```

```

In [16]: # we use seperated optimizers for training generator and discriminator
generator_optimizer = tf.keras.optimizers.Adam(hparas['LR'])
discriminator_optimizer = tf.keras.optimizers.Adam(hparas['LR'])

```

```

In [17]: # one benefit of tf.train.Checkpoint() API is we can save everything seperately
checkpoint_dir = hparas['CHECKPOINTS_DIR']
checkpoint_prefix = os.path.join(checkpoint_dir, "ckpt")
checkpoint = tf.train.Checkpoint(generator_optimizer=generator_optimizer,
                                discriminator_optimizer=discriminator_optimizer,
                                text_encoder=text_encoder,
                                generator=generator,
                                discriminator=discriminator)

```

```

In [18]: @tf.function
def train_step(real_image, caption, hidden):
    # random noise for generator
    noise = tf.random.normal(shape=[hparas['BATCH_SIZE'], hparas['Z_DIM']], mean=0.0, st

    with tf.GradientTape() as gen_tape, tf.GradientTape() as disc_tape:
        text_embed, hidden = text_encoder(caption, hidden)
        _, fake_image = generator(text_embed, noise)
        real_logits, real_output = discriminator(real_image, text_embed)
        fake_logits, fake_output = discriminator(fake_image, text_embed)

        g_loss = generator_loss(fake_logits)
        d_loss = discriminator_loss(real_logits, fake_logits)

        grad_g = gen_tape.gradient(g_loss, generator.trainable_variables)
        grad_d = disc_tape.gradient(d_loss, discriminator.trainable_variables)

        generator_optimizer.apply_gradients(zip(grad_g, generator.trainable_variables))
        discriminator_optimizer.apply_gradients(zip(grad_d, discriminator.trainable_variables))

    return g_loss, d_loss

```

```

In [19]: @tf.function
def test_step(caption, noise, hidden):
    text_embed, hidden = text_encoder(caption, hidden)
    _, fake_image = generator(text_embed, noise)
    return fake_image

```

## Visualiztion

During training, we can visualize the generated image to evaluate the quality of generator. The followings are some functions helping visualization.

```

In [20]: def merge(images, size):
h, w = images.shape[1], images.shape[2]
img = np.zeros((h * size[0], w * size[1], 3))
for idx, image in enumerate(images):
    i = idx % size[1]

```

```

        j = idx // size[1]
        img[j*h:j*h+h, i*w:i*w+w, :] = image
    return img

def imsave(images, size, path):
    # getting the pixel values between [0, 1] to save it
    return plt.imsave(path, merge(images, size)*0.5 + 0.5)

def save_images(images, size, image_path):
    return imsave(images, size, image_path)

```

```

In [21]: def sample_generator(caption, batch_size):
        caption = np.asarray(caption)
        caption = caption.astype(np.int)
        dataset = tf.data.Dataset.from_tensor_slices(caption)
        dataset = dataset.batch(batch_size)
        return dataset

```

We always use same random seed and same sentences during training, which is more convenient for us to evaluate the quality of generated image.

```

In [22]: ni = int(np.ceil(np.sqrt(hparas['BATCH_SIZE'])))
        sample_size = hparas['BATCH_SIZE']
        sample_seed = np.random.normal(loc=0.0, scale=1.0, size=(sample_size, hparas['Z_DIM'])).
        sample_sentence = ["the flower shown has yellow anther red pistil and bright red petals.
                            ["this flower has petals that are yellow, white and purple and has dar
                            ["the petals on this flower are white with a yellow center"] * int(sam
                            ["this flower has a lot of small round pink petals."] * int(sample_siz
                            ["this flower is orange in color, and has petals that are ruffled and
                            ["the flower has yellow petals and the center of it is brown."] * int(
                            ["this flower has petals that are blue and white."] * int(sample_size/
                            ["these white flowers have petals that start off white in color and en

        for i, sent in enumerate(sample_sentence):
            sample_sentence[i] = sent2IdList(sent)
        sample_sentence = sample_generator(sample_sentence, hparas['BATCH_SIZE'])

```

/tmp/ipykernel\_29626/646496249.py:3: DeprecationWarning: `np.int` is a deprecated alias for the builtin `int`. To silence this warning, use `int` by itself. Doing this will not modify any behavior and is safe. When replacing `np.int`, you may wish to use e.g. `np.int64` or `np.int32` to specify the precision. If you wish to review your current use, check the release note link for additional information.  
Deprecated in NumPy 1.20; for more details and guidance: <https://numpy.org/devdocs/release/1.20.0-notes.html#deprecations>

```
caption = caption.astype(np.int)
```

## Training

```

In [23]: if not os.path.exists('samples/demo'):
        os.makedirs('samples/demo')

```

```

In [24]: def train(dataset, epochs):
        # hidden state of RNN
        hidden = text_encoder.initialize_hidden_state()
        steps_per_epoch = int(hparas['N_SAMPLE']/hparas['BATCH_SIZE'])

        for epoch in range(hparas['N_EPOCH']):
            g_total_loss = 0
            d_total_loss = 0
            start = time.time()

```

```

for image, caption in dataset:
    g_loss, d_loss = train_step(image, caption, hidden)
    g_total_loss += g_loss
    d_total_loss += d_loss

time_tuple = time.localtime()
time_string = time.strftime("%m/%d/%Y, %H:%M:%S", time_tuple)

print("Epoch {}, gen_loss: {:.4f}, disc_loss: {:.4f}".format(epoch+1,
                                                             g_total_loss/steps_
                                                             d_total_loss/steps_

print('Time for epoch {} is {:.4f} sec'.format(epoch+1, time.time()-start))

# save the model
if (epoch + 1) % 50 == 0:
    checkpoint.save(file_prefix = checkpoint_prefix)

# visualization
if (epoch + 1) % hparas['PRINT_FREQ'] == 0:
    for caption in sample_sentence:
        fake_image = test_step(caption, sample_seed, hidden)
        save_images(fake_image, [ni, ni], 'samples/demo/train_{:02d}.jpg'.format(epc

```

In [25]: train(dataset, hparas['N\_EPOCH'])

2022-11-23 10:51:06.854118: I tensorflow/stream\_executor/cuda/cuda\_dnn.cc:384] Loaded cuDNN version 8201

```

Epoch 1, gen_loss: 0.4364, disc_loss: 1.1589
Time for epoch 1 is 6.4657 sec
Epoch 2, gen_loss: 0.4609, disc_loss: 1.1169
Time for epoch 2 is 2.8109 sec
Epoch 3, gen_loss: 0.7003, disc_loss: 0.8476
Time for epoch 3 is 2.8191 sec
Epoch 4, gen_loss: 1.1180, disc_loss: 0.5337
Time for epoch 4 is 2.9703 sec
Epoch 5, gen_loss: 2.1168, disc_loss: 0.1785
Time for epoch 5 is 2.8329 sec
Epoch 6, gen_loss: 1.9798, disc_loss: 0.2144
Time for epoch 6 is 2.8044 sec
Epoch 7, gen_loss: 2.2416, disc_loss: 0.1638
Time for epoch 7 is 2.7629 sec
Epoch 8, gen_loss: 2.6708, disc_loss: 0.1144
Time for epoch 8 is 2.7492 sec
Epoch 9, gen_loss: 2.7469, disc_loss: 0.1200
Time for epoch 9 is 2.7638 sec
Epoch 10, gen_loss: 3.0936, disc_loss: 0.1244
Time for epoch 10 is 2.5924 sec
Epoch 11, gen_loss: 3.5398, disc_loss: 0.1227
Time for epoch 11 is 2.5874 sec
Epoch 12, gen_loss: 3.2764, disc_loss: 0.2010
Time for epoch 12 is 2.8076 sec
Epoch 13, gen_loss: 3.7002, disc_loss: 0.2672
Time for epoch 13 is 2.7153 sec
Epoch 14, gen_loss: 3.8901, disc_loss: 0.3380
Time for epoch 14 is 2.6933 sec
Epoch 15, gen_loss: 3.3203, disc_loss: 0.4580
Time for epoch 15 is 2.7301 sec
Epoch 16, gen_loss: 3.2107, disc_loss: 0.5796
Time for epoch 16 is 2.7159 sec
Epoch 17, gen_loss: 3.3180, disc_loss: 0.5807
Time for epoch 17 is 2.8851 sec
Epoch 18, gen_loss: 3.2794, disc_loss: 0.5899

```

Time for epoch 18 is 2.8288 sec  
Epoch 19, gen\_loss: 2.9915, disc\_loss: 0.6637  
Time for epoch 19 is 2.6269 sec  
Epoch 20, gen\_loss: 2.8574, disc\_loss: 0.7690  
Time for epoch 20 is 2.9169 sec  
Epoch 21, gen\_loss: 2.7805, disc\_loss: 0.6750  
Time for epoch 21 is 2.7499 sec  
Epoch 22, gen\_loss: 2.4794, disc\_loss: 0.7101  
Time for epoch 22 is 2.7760 sec  
Epoch 23, gen\_loss: 2.5188, disc\_loss: 0.6775  
Time for epoch 23 is 2.8413 sec  
Epoch 24, gen\_loss: 2.2181, disc\_loss: 0.7694  
Time for epoch 24 is 2.8376 sec  
Epoch 25, gen\_loss: 2.0179, disc\_loss: 0.8207  
Time for epoch 25 is 2.7066 sec  
Epoch 26, gen\_loss: 1.9934, disc\_loss: 0.8338  
Time for epoch 26 is 2.7065 sec  
Epoch 27, gen\_loss: 2.2491, disc\_loss: 0.8071  
Time for epoch 27 is 2.7918 sec  
Epoch 28, gen\_loss: 2.2550, disc\_loss: 0.7524  
Time for epoch 28 is 2.7575 sec  
Epoch 29, gen\_loss: 2.4824, disc\_loss: 0.5772  
Time for epoch 29 is 2.8193 sec  
Epoch 30, gen\_loss: 2.4119, disc\_loss: 0.5899  
Time for epoch 30 is 2.7940 sec  
Epoch 31, gen\_loss: 2.2011, disc\_loss: 0.6949  
Time for epoch 31 is 2.7396 sec  
Epoch 32, gen\_loss: 2.4667, disc\_loss: 0.5465  
Time for epoch 32 is 2.8088 sec  
Epoch 33, gen\_loss: 2.2277, disc\_loss: 0.5884  
Time for epoch 33 is 2.8558 sec  
Epoch 34, gen\_loss: 1.8392, disc\_loss: 0.7087  
Time for epoch 34 is 2.6343 sec  
Epoch 35, gen\_loss: 1.6541, disc\_loss: 0.8404  
Time for epoch 35 is 2.6432 sec  
Epoch 36, gen\_loss: 1.7993, disc\_loss: 0.7966  
Time for epoch 36 is 2.8168 sec  
Epoch 37, gen\_loss: 1.5980, disc\_loss: 0.8697  
Time for epoch 37 is 2.8117 sec  
Epoch 38, gen\_loss: 1.5760, disc\_loss: 0.8208  
Time for epoch 38 is 2.7579 sec  
Epoch 39, gen\_loss: 1.5320, disc\_loss: 0.7401  
Time for epoch 39 is 2.9055 sec  
Epoch 40, gen\_loss: 1.3264, disc\_loss: 0.8475  
Time for epoch 40 is 2.8203 sec  
Epoch 41, gen\_loss: 1.3784, disc\_loss: 0.8276  
Time for epoch 41 is 2.6340 sec  
Epoch 42, gen\_loss: 1.2372, disc\_loss: 0.9461  
Time for epoch 42 is 2.6340 sec  
Epoch 43, gen\_loss: 1.1619, disc\_loss: 1.0563  
Time for epoch 43 is 2.6973 sec  
Epoch 44, gen\_loss: 1.2657, disc\_loss: 0.9792  
Time for epoch 44 is 2.5737 sec  
Epoch 45, gen\_loss: 1.3325, disc\_loss: 0.9950  
Time for epoch 45 is 2.6825 sec  
Epoch 46, gen\_loss: 1.1811, disc\_loss: 1.1596  
Time for epoch 46 is 2.6573 sec  
Epoch 47, gen\_loss: 1.2318, disc\_loss: 1.1932  
Time for epoch 47 is 2.6589 sec  
Epoch 48, gen\_loss: 1.3921, disc\_loss: 1.1612  
Time for epoch 48 is 2.8030 sec  
Epoch 49, gen\_loss: 1.8097, disc\_loss: 0.8338  
Time for epoch 49 is 2.6953 sec

Epoch 50, gen\_loss: 1.9475, disc\_loss: 0.8069  
Time for epoch 50 is 2.8073 sec  
Epoch 51, gen\_loss: 2.0441, disc\_loss: 0.7018  
Time for epoch 51 is 2.8398 sec  
Epoch 52, gen\_loss: 1.7792, disc\_loss: 0.7774  
Time for epoch 52 is 2.6281 sec  
Epoch 53, gen\_loss: 1.8446, disc\_loss: 0.6930  
Time for epoch 53 is 2.6442 sec  
Epoch 54, gen\_loss: 1.3656, disc\_loss: 0.9966  
Time for epoch 54 is 2.9931 sec  
Epoch 55, gen\_loss: 1.3377, disc\_loss: 1.0219  
Time for epoch 55 is 2.9101 sec  
Epoch 56, gen\_loss: 1.3888, disc\_loss: 0.9737  
Time for epoch 56 is 2.9658 sec  
Epoch 57, gen\_loss: 1.4115, disc\_loss: 1.0109  
Time for epoch 57 is 2.6824 sec  
Epoch 58, gen\_loss: 1.5952, disc\_loss: 0.9119  
Time for epoch 58 is 2.6289 sec  
Epoch 59, gen\_loss: 1.4854, disc\_loss: 0.9072  
Time for epoch 59 is 2.7560 sec  
Epoch 60, gen\_loss: 1.5935, disc\_loss: 0.8635  
Time for epoch 60 is 2.7365 sec  
Epoch 61, gen\_loss: 1.7238, disc\_loss: 0.8577  
Time for epoch 61 is 2.6844 sec  
Epoch 62, gen\_loss: 1.5285, disc\_loss: 0.9000  
Time for epoch 62 is 2.5478 sec  
Epoch 63, gen\_loss: 2.0553, disc\_loss: 0.6168  
Time for epoch 63 is 2.6852 sec  
Epoch 64, gen\_loss: 1.5557, disc\_loss: 0.8291  
Time for epoch 64 is 2.7248 sec  
Epoch 65, gen\_loss: 1.4671, disc\_loss: 0.8832  
Time for epoch 65 is 2.7288 sec  
Epoch 66, gen\_loss: 1.4583, disc\_loss: 0.9236  
Time for epoch 66 is 2.8655 sec  
Epoch 67, gen\_loss: 1.2149, disc\_loss: 1.1991  
Time for epoch 67 is 2.8211 sec  
Epoch 68, gen\_loss: 1.4782, disc\_loss: 0.9738  
Time for epoch 68 is 2.6730 sec  
Epoch 69, gen\_loss: 1.5135, disc\_loss: 0.9301  
Time for epoch 69 is 2.6890 sec  
Epoch 70, gen\_loss: 1.6259, disc\_loss: 0.9133  
Time for epoch 70 is 2.7414 sec  
Epoch 71, gen\_loss: 1.6113, disc\_loss: 0.8651  
Time for epoch 71 is 2.8060 sec  
Epoch 72, gen\_loss: 1.5670, disc\_loss: 0.9376  
Time for epoch 72 is 2.7964 sec  
Epoch 73, gen\_loss: 1.6260, disc\_loss: 0.7706  
Time for epoch 73 is 2.7392 sec  
Epoch 74, gen\_loss: 1.5159, disc\_loss: 0.8097  
Time for epoch 74 is 2.7193 sec  
Epoch 75, gen\_loss: 1.4416, disc\_loss: 0.9139  
Time for epoch 75 is 2.6357 sec  
Epoch 76, gen\_loss: 1.3253, disc\_loss: 1.0042  
Time for epoch 76 is 2.7168 sec  
Epoch 77, gen\_loss: 1.4751, disc\_loss: 0.9114  
Time for epoch 77 is 2.9280 sec  
Epoch 78, gen\_loss: 1.1866, disc\_loss: 1.1838  
Time for epoch 78 is 2.7866 sec  
Epoch 79, gen\_loss: 1.4802, disc\_loss: 0.9979  
Time for epoch 79 is 2.8527 sec  
Epoch 80, gen\_loss: 1.3509, disc\_loss: 1.1450  
Time for epoch 80 is 2.8522 sec  
Epoch 81, gen\_loss: 1.5979, disc\_loss: 0.9789

Time for epoch 81 is 2.7772 sec  
Epoch 82, gen\_loss: 1.6496, disc\_loss: 0.8433  
Time for epoch 82 is 2.7292 sec  
Epoch 83, gen\_loss: 1.8771, disc\_loss: 0.7104  
Time for epoch 83 is 2.7698 sec  
Epoch 84, gen\_loss: 1.5543, disc\_loss: 0.9779  
Time for epoch 84 is 2.8987 sec  
Epoch 85, gen\_loss: 1.6199, disc\_loss: 0.8222  
Time for epoch 85 is 2.6661 sec  
Epoch 86, gen\_loss: 1.3985, disc\_loss: 0.9203  
Time for epoch 86 is 2.7393 sec  
Epoch 87, gen\_loss: 1.3435, disc\_loss: 0.9956  
Time for epoch 87 is 2.8992 sec  
Epoch 88, gen\_loss: 1.3395, disc\_loss: 1.0460  
Time for epoch 88 is 2.6905 sec  
Epoch 89, gen\_loss: 1.1854, disc\_loss: 1.1986  
Time for epoch 89 is 2.7423 sec  
Epoch 90, gen\_loss: 1.2187, disc\_loss: 1.2287  
Time for epoch 90 is 2.7155 sec  
Epoch 91, gen\_loss: 1.6799, disc\_loss: 0.9987  
Time for epoch 91 is 2.6935 sec  
Epoch 92, gen\_loss: 1.5032, disc\_loss: 1.0959  
Time for epoch 92 is 2.6835 sec  
Epoch 93, gen\_loss: 1.8747, disc\_loss: 0.7749  
Time for epoch 93 is 2.6922 sec  
Epoch 94, gen\_loss: 1.8685, disc\_loss: 0.7463  
Time for epoch 94 is 2.8375 sec  
Epoch 95, gen\_loss: 1.5439, disc\_loss: 0.9151  
Time for epoch 95 is 2.5907 sec  
Epoch 96, gen\_loss: 1.6795, disc\_loss: 0.9026  
Time for epoch 96 is 2.6272 sec  
Epoch 97, gen\_loss: 1.6810, disc\_loss: 0.8357  
Time for epoch 97 is 2.8076 sec  
Epoch 98, gen\_loss: 1.2692, disc\_loss: 1.1110  
Time for epoch 98 is 2.7257 sec  
Epoch 99, gen\_loss: 1.5281, disc\_loss: 0.9534  
Time for epoch 99 is 2.7764 sec  
Epoch 100, gen\_loss: 1.3780, disc\_loss: 1.0605  
Time for epoch 100 is 2.7014 sec  
Epoch 101, gen\_loss: 1.2994, disc\_loss: 1.0896  
Time for epoch 101 is 2.7676 sec  
Epoch 102, gen\_loss: 1.3778, disc\_loss: 1.0861  
Time for epoch 102 is 2.5861 sec  
Epoch 103, gen\_loss: 1.4424, disc\_loss: 1.0795  
Time for epoch 103 is 2.7135 sec  
Epoch 104, gen\_loss: 1.6152, disc\_loss: 0.9349  
Time for epoch 104 is 2.6470 sec  
Epoch 105, gen\_loss: 1.5634, disc\_loss: 0.8363  
Time for epoch 105 is 2.6651 sec  
Epoch 106, gen\_loss: 1.5207, disc\_loss: 0.8634  
Time for epoch 106 is 2.5486 sec  
Epoch 107, gen\_loss: 1.4024, disc\_loss: 1.0777  
Time for epoch 107 is 2.7026 sec  
Epoch 108, gen\_loss: 1.4811, disc\_loss: 1.0022  
Time for epoch 108 is 2.7292 sec  
Epoch 109, gen\_loss: 1.3014, disc\_loss: 1.1267  
Time for epoch 109 is 2.7041 sec  
Epoch 110, gen\_loss: 1.7332, disc\_loss: 0.9028  
Time for epoch 110 is 2.5589 sec  
Epoch 111, gen\_loss: 1.6340, disc\_loss: 0.9368  
Time for epoch 111 is 2.6403 sec  
Epoch 112, gen\_loss: 1.7182, disc\_loss: 0.8618  
Time for epoch 112 is 2.6824 sec

Epoch 113, gen\_loss: 1.7934, disc\_loss: 0.8071  
Time for epoch 113 is 2.6463 sec  
Epoch 114, gen\_loss: 1.6865, disc\_loss: 0.9140  
Time for epoch 114 is 2.8104 sec  
Epoch 115, gen\_loss: 1.5720, disc\_loss: 0.8998  
Time for epoch 115 is 2.7755 sec  
Epoch 116, gen\_loss: 1.4464, disc\_loss: 0.9584  
Time for epoch 116 is 2.7589 sec  
Epoch 117, gen\_loss: 1.4378, disc\_loss: 1.0738  
Time for epoch 117 is 2.7641 sec  
Epoch 118, gen\_loss: 1.4586, disc\_loss: 0.9319  
Time for epoch 118 is 2.7699 sec  
Epoch 119, gen\_loss: 1.2176, disc\_loss: 1.0874  
Time for epoch 119 is 2.7896 sec  
Epoch 120, gen\_loss: 1.3419, disc\_loss: 1.0732  
Time for epoch 120 is 2.6355 sec  
Epoch 121, gen\_loss: 1.3132, disc\_loss: 1.0574  
Time for epoch 121 is 2.7735 sec  
Epoch 122, gen\_loss: 1.1665, disc\_loss: 1.2653  
Time for epoch 122 is 2.8039 sec  
Epoch 123, gen\_loss: 1.2515, disc\_loss: 1.1753  
Time for epoch 123 is 2.7389 sec  
Epoch 124, gen\_loss: 1.4718, disc\_loss: 1.0452  
Time for epoch 124 is 2.6802 sec  
Epoch 125, gen\_loss: 1.5014, disc\_loss: 1.0212  
Time for epoch 125 is 2.6490 sec  
Epoch 126, gen\_loss: 1.6835, disc\_loss: 0.8953  
Time for epoch 126 is 2.7503 sec  
Epoch 127, gen\_loss: 1.7121, disc\_loss: 0.9067  
Time for epoch 127 is 2.7326 sec  
Epoch 128, gen\_loss: 1.7355, disc\_loss: 0.9228  
Time for epoch 128 is 2.6401 sec  
Epoch 129, gen\_loss: 1.4000, disc\_loss: 1.0729  
Time for epoch 129 is 2.6908 sec  
Epoch 130, gen\_loss: 1.5924, disc\_loss: 0.9089  
Time for epoch 130 is 2.7549 sec  
Epoch 131, gen\_loss: 1.3808, disc\_loss: 1.0266  
Time for epoch 131 is 2.7874 sec  
Epoch 132, gen\_loss: 1.3226, disc\_loss: 1.0729  
Time for epoch 132 is 2.6691 sec  
Epoch 133, gen\_loss: 1.4847, disc\_loss: 1.0205  
Time for epoch 133 is 2.7808 sec  
Epoch 134, gen\_loss: 1.3367, disc\_loss: 1.1488  
Time for epoch 134 is 2.6593 sec  
Epoch 135, gen\_loss: 1.3471, disc\_loss: 1.1402  
Time for epoch 135 is 2.6591 sec  
Epoch 136, gen\_loss: 1.3646, disc\_loss: 1.0708  
Time for epoch 136 is 2.6642 sec  
Epoch 137, gen\_loss: 1.4698, disc\_loss: 0.9890  
Time for epoch 137 is 2.6439 sec  
Epoch 138, gen\_loss: 1.4207, disc\_loss: 1.1021  
Time for epoch 138 is 2.6299 sec  
Epoch 139, gen\_loss: 1.5090, disc\_loss: 0.9443  
Time for epoch 139 is 2.8363 sec  
Epoch 140, gen\_loss: 1.6061, disc\_loss: 0.9085  
Time for epoch 140 is 2.7644 sec  
Epoch 141, gen\_loss: 1.4787, disc\_loss: 1.0390  
Time for epoch 141 is 2.7733 sec  
Epoch 142, gen\_loss: 1.5669, disc\_loss: 0.9498  
Time for epoch 142 is 2.8179 sec  
Epoch 143, gen\_loss: 1.4865, disc\_loss: 1.0280  
Time for epoch 143 is 2.8063 sec  
Epoch 144, gen\_loss: 1.3309, disc\_loss: 1.1425



Time for epoch 144 is 2.7698 sec  
Epoch 145, gen\_loss: 1.6407, disc\_loss: 0.9441  
Time for epoch 145 is 2.7685 sec  
Epoch 146, gen\_loss: 1.6000, disc\_loss: 1.0913  
Time for epoch 146 is 2.6512 sec  
Epoch 147, gen\_loss: 1.5675, disc\_loss: 1.0543  
Time for epoch 147 is 2.7228 sec  
Epoch 148, gen\_loss: 1.9242, disc\_loss: 0.7131  
Time for epoch 148 is 2.5541 sec  
Epoch 149, gen\_loss: 1.4638, disc\_loss: 0.9551  
Time for epoch 149 is 2.6536 sec  
Epoch 150, gen\_loss: 1.4844, disc\_loss: 0.9487  
Time for epoch 150 is 2.5515 sec  
Epoch 151, gen\_loss: 1.3229, disc\_loss: 1.0601  
Time for epoch 151 is 2.8130 sec  
Epoch 152, gen\_loss: 1.4318, disc\_loss: 0.9633  
Time for epoch 152 is 2.7064 sec  
Epoch 153, gen\_loss: 1.4065, disc\_loss: 0.9805  
Time for epoch 153 is 2.7590 sec  
Epoch 154, gen\_loss: 1.3065, disc\_loss: 1.1302  
Time for epoch 154 is 2.7382 sec  
Epoch 155, gen\_loss: 1.3249, disc\_loss: 1.1022  
Time for epoch 155 is 2.7275 sec  
Epoch 156, gen\_loss: 1.5262, disc\_loss: 1.0262  
Time for epoch 156 is 2.6433 sec  
Epoch 157, gen\_loss: 1.1472, disc\_loss: 1.3655  
Time for epoch 157 is 2.6548 sec  
Epoch 158, gen\_loss: 1.3324, disc\_loss: 1.1970  
Time for epoch 158 is 2.7177 sec  
Epoch 159, gen\_loss: 1.8308, disc\_loss: 0.9318  
Time for epoch 159 is 2.7238 sec  
Epoch 160, gen\_loss: 1.2781, disc\_loss: 1.2044  
Time for epoch 160 is 2.7112 sec  
Epoch 161, gen\_loss: 1.6296, disc\_loss: 1.0205  
Time for epoch 161 is 2.6859 sec  
Epoch 162, gen\_loss: 1.7293, disc\_loss: 0.8931  
Time for epoch 162 is 2.8377 sec  
Epoch 163, gen\_loss: 1.4117, disc\_loss: 1.1352  
Time for epoch 163 is 2.8510 sec  
Epoch 164, gen\_loss: 1.5326, disc\_loss: 1.0279  
Time for epoch 164 is 2.6824 sec  
Epoch 165, gen\_loss: 1.7529, disc\_loss: 0.8632  
Time for epoch 165 is 2.6333 sec  
Epoch 166, gen\_loss: 1.2982, disc\_loss: 1.1294  
Time for epoch 166 is 2.7312 sec  
Epoch 167, gen\_loss: 1.4800, disc\_loss: 0.9923  
Time for epoch 167 is 2.7350 sec  
Epoch 168, gen\_loss: 1.4480, disc\_loss: 1.0499  
Time for epoch 168 is 2.6698 sec  
Epoch 169, gen\_loss: 1.4168, disc\_loss: 0.9776  
Time for epoch 169 is 2.6832 sec  
Epoch 170, gen\_loss: 1.2764, disc\_loss: 1.1027  
Time for epoch 170 is 2.6964 sec  
Epoch 171, gen\_loss: 1.2168, disc\_loss: 1.1893  
Time for epoch 171 is 2.7892 sec  
Epoch 172, gen\_loss: 1.2654, disc\_loss: 1.2524  
Time for epoch 172 is 2.7115 sec  
Epoch 173, gen\_loss: 1.4095, disc\_loss: 1.1211  
Time for epoch 173 is 2.7019 sec  
Epoch 174, gen\_loss: 1.4555, disc\_loss: 1.2201  
Time for epoch 174 is 2.7992 sec  
Epoch 175, gen\_loss: 1.6975, disc\_loss: 0.9438  
Time for epoch 175 is 2.7630 sec

Epoch 176, gen\_loss: 1.4668, disc\_loss: 1.0841  
Time for epoch 176 is 2.7659 sec  
Epoch 177, gen\_loss: 1.6393, disc\_loss: 0.9737  
Time for epoch 177 is 2.9364 sec  
Epoch 178, gen\_loss: 1.3379, disc\_loss: 1.1710  
Time for epoch 178 is 2.5878 sec  
Epoch 179, gen\_loss: 1.5771, disc\_loss: 1.0915  
Time for epoch 179 is 2.7223 sec  
Epoch 180, gen\_loss: 1.6919, disc\_loss: 0.9597  
Time for epoch 180 is 2.6808 sec  
Epoch 181, gen\_loss: 1.4670, disc\_loss: 1.1084  
Time for epoch 181 is 2.7678 sec  
Epoch 182, gen\_loss: 1.6527, disc\_loss: 1.0418  
Time for epoch 182 is 2.6347 sec  
Epoch 183, gen\_loss: 1.5148, disc\_loss: 1.1753  
Time for epoch 183 is 2.7407 sec  
Epoch 184, gen\_loss: 1.7776, disc\_loss: 0.9156  
Time for epoch 184 is 2.6605 sec  
Epoch 185, gen\_loss: 1.6029, disc\_loss: 0.9785  
Time for epoch 185 is 2.6741 sec  
Epoch 186, gen\_loss: 1.7658, disc\_loss: 0.7776  
Time for epoch 186 is 2.7111 sec  
Epoch 187, gen\_loss: 1.4713, disc\_loss: 0.9678  
Time for epoch 187 is 2.7466 sec  
Epoch 188, gen\_loss: 1.5302, disc\_loss: 0.9258  
Time for epoch 188 is 2.8551 sec  
Epoch 189, gen\_loss: 1.2274, disc\_loss: 1.1876  
Time for epoch 189 is 2.6145 sec  
Epoch 190, gen\_loss: 1.2406, disc\_loss: 1.1040  
Time for epoch 190 is 2.7718 sec  
Epoch 191, gen\_loss: 1.2513, disc\_loss: 1.2083  
Time for epoch 191 is 2.8533 sec  
Epoch 192, gen\_loss: 1.2350, disc\_loss: 1.2202  
Time for epoch 192 is 2.7032 sec  
Epoch 193, gen\_loss: 1.3220, disc\_loss: 1.1078  
Time for epoch 193 is 2.8181 sec  
Epoch 194, gen\_loss: 1.2431, disc\_loss: 1.4220  
Time for epoch 194 is 2.8269 sec  
Epoch 195, gen\_loss: 1.1901, disc\_loss: 1.3847  
Time for epoch 195 is 2.8127 sec  
Epoch 196, gen\_loss: 1.5008, disc\_loss: 1.0429  
Time for epoch 196 is 2.7738 sec  
Epoch 197, gen\_loss: 1.3847, disc\_loss: 1.1408  
Time for epoch 197 is 2.8744 sec  
Epoch 198, gen\_loss: 1.5579, disc\_loss: 1.1464  
Time for epoch 198 is 2.7678 sec  
Epoch 199, gen\_loss: 1.5096, disc\_loss: 1.2377  
Time for epoch 199 is 2.8698 sec  
Epoch 200, gen\_loss: 1.4926, disc\_loss: 1.1002  
Time for epoch 200 is 2.7252 sec  
Epoch 201, gen\_loss: 1.8154, disc\_loss: 0.9880  
Time for epoch 201 is 2.6172 sec  
Epoch 202, gen\_loss: 1.8034, disc\_loss: 0.8771  
Time for epoch 202 is 2.6201 sec  
Epoch 203, gen\_loss: 1.5226, disc\_loss: 1.0093  
Time for epoch 203 is 2.8317 sec  
Epoch 204, gen\_loss: 1.4101, disc\_loss: 1.0743  
Time for epoch 204 is 2.6827 sec  
Epoch 205, gen\_loss: 1.6045, disc\_loss: 0.9989  
Time for epoch 205 is 2.7057 sec  
Epoch 206, gen\_loss: 1.5377, disc\_loss: 1.0190  
Time for epoch 206 is 2.7033 sec  
Epoch 207, gen\_loss: 1.5059, disc\_loss: 1.0256

Time for epoch 207 is 2.7224 sec  
Epoch 208, gen\_loss: 1.4472, disc\_loss: 1.0772  
Time for epoch 208 is 2.7084 sec  
Epoch 209, gen\_loss: 1.4207, disc\_loss: 1.1413  
Time for epoch 209 is 2.7778 sec  
Epoch 210, gen\_loss: 1.3966, disc\_loss: 1.1686  
Time for epoch 210 is 2.6917 sec  
Epoch 211, gen\_loss: 1.6718, disc\_loss: 0.9500  
Time for epoch 211 is 2.6727 sec  
Epoch 212, gen\_loss: 1.4375, disc\_loss: 1.2049  
Time for epoch 212 is 2.6813 sec  
Epoch 213, gen\_loss: 1.5637, disc\_loss: 1.1077  
Time for epoch 213 is 2.8819 sec  
Epoch 214, gen\_loss: 1.6571, disc\_loss: 1.0925  
Time for epoch 214 is 2.9648 sec  
Epoch 215, gen\_loss: 1.7018, disc\_loss: 1.0251  
Time for epoch 215 is 2.9791 sec  
Epoch 216, gen\_loss: 1.8695, disc\_loss: 0.8181  
Time for epoch 216 is 2.8715 sec  
Epoch 217, gen\_loss: 1.7698, disc\_loss: 0.7700  
Time for epoch 217 is 2.8623 sec  
Epoch 218, gen\_loss: 1.6875, disc\_loss: 0.8059  
Time for epoch 218 is 2.7885 sec  
Epoch 219, gen\_loss: 1.5561, disc\_loss: 0.9675  
Time for epoch 219 is 2.6436 sec  
Epoch 220, gen\_loss: 1.4371, disc\_loss: 1.0898  
Time for epoch 220 is 2.6830 sec  
Epoch 221, gen\_loss: 1.3833, disc\_loss: 1.1408  
Time for epoch 221 is 2.7298 sec  
Epoch 222, gen\_loss: 1.4868, disc\_loss: 1.1606  
Time for epoch 222 is 2.6984 sec  
Epoch 223, gen\_loss: 1.4988, disc\_loss: 1.0247  
Time for epoch 223 is 2.6416 sec  
Epoch 224, gen\_loss: 1.3950, disc\_loss: 1.0279  
Time for epoch 224 is 2.7603 sec  
Epoch 225, gen\_loss: 1.2873, disc\_loss: 1.1358  
Time for epoch 225 is 2.7235 sec  
Epoch 226, gen\_loss: 1.3522, disc\_loss: 1.1329  
Time for epoch 226 is 2.7960 sec  
Epoch 227, gen\_loss: 1.0637, disc\_loss: 1.4348  
Time for epoch 227 is 2.9057 sec  
Epoch 228, gen\_loss: 1.4231, disc\_loss: 1.1979  
Time for epoch 228 is 2.8483 sec  
Epoch 229, gen\_loss: 1.5613, disc\_loss: 1.1716  
Time for epoch 229 is 2.6304 sec  
Epoch 230, gen\_loss: 1.4185, disc\_loss: 1.2045  
Time for epoch 230 is 2.8518 sec  
Epoch 231, gen\_loss: 1.6212, disc\_loss: 1.0430  
Time for epoch 231 is 2.7309 sec  
Epoch 232, gen\_loss: 1.7648, disc\_loss: 1.0391  
Time for epoch 232 is 2.6946 sec  
Epoch 233, gen\_loss: 1.5442, disc\_loss: 1.2060  
Time for epoch 233 is 2.8439 sec  
Epoch 234, gen\_loss: 1.8573, disc\_loss: 0.9331  
Time for epoch 234 is 2.8349 sec  
Epoch 235, gen\_loss: 2.0443, disc\_loss: 0.8022  
Time for epoch 235 is 2.9737 sec  
Epoch 236, gen\_loss: 1.9171, disc\_loss: 0.7502  
Time for epoch 236 is 2.7987 sec  
Epoch 237, gen\_loss: 1.9386, disc\_loss: 0.7525  
Time for epoch 237 is 2.6919 sec  
Epoch 238, gen\_loss: 1.5386, disc\_loss: 0.9561  
Time for epoch 238 is 2.7903 sec

Epoch 239, gen\_loss: 1.4744, disc\_loss: 1.0515  
Time for epoch 239 is 2.7831 sec  
Epoch 240, gen\_loss: 1.4129, disc\_loss: 1.1608  
Time for epoch 240 is 2.6549 sec  
Epoch 241, gen\_loss: 1.3050, disc\_loss: 1.3024  
Time for epoch 241 is 2.9748 sec  
Epoch 242, gen\_loss: 1.3427, disc\_loss: 1.3079  
Time for epoch 242 is 2.8496 sec  
Epoch 243, gen\_loss: 1.3053, disc\_loss: 1.3177  
Time for epoch 243 is 2.7616 sec  
Epoch 244, gen\_loss: 1.7363, disc\_loss: 0.9112  
Time for epoch 244 is 2.7450 sec  
Epoch 245, gen\_loss: 1.3155, disc\_loss: 1.2216  
Time for epoch 245 is 2.6733 sec  
Epoch 246, gen\_loss: 1.4095, disc\_loss: 1.1270  
Time for epoch 246 is 2.7029 sec  
Epoch 247, gen\_loss: 1.5622, disc\_loss: 1.1004  
Time for epoch 247 is 2.7613 sec  
Epoch 248, gen\_loss: 1.7007, disc\_loss: 0.9649  
Time for epoch 248 is 2.9177 sec  
Epoch 249, gen\_loss: 1.3659, disc\_loss: 1.2208  
Time for epoch 249 is 2.7960 sec  
Epoch 250, gen\_loss: 1.9298, disc\_loss: 0.8285  
Time for epoch 250 is 2.7713 sec  
Epoch 251, gen\_loss: 1.7595, disc\_loss: 0.8405  
Time for epoch 251 is 2.8186 sec  
Epoch 252, gen\_loss: 1.5344, disc\_loss: 1.0992  
Time for epoch 252 is 2.7067 sec  
Epoch 253, gen\_loss: 1.6915, disc\_loss: 0.9595  
Time for epoch 253 is 2.7990 sec  
Epoch 254, gen\_loss: 1.5983, disc\_loss: 1.0121  
Time for epoch 254 is 2.8418 sec  
Epoch 255, gen\_loss: 1.3506, disc\_loss: 1.1591  
Time for epoch 255 is 2.7906 sec  
Epoch 256, gen\_loss: 1.5117, disc\_loss: 1.0851  
Time for epoch 256 is 2.7844 sec  
Epoch 257, gen\_loss: 1.3142, disc\_loss: 1.2615  
Time for epoch 257 is 2.8266 sec  
Epoch 258, gen\_loss: 1.4411, disc\_loss: 1.0700  
Time for epoch 258 is 2.7264 sec  
Epoch 259, gen\_loss: 1.1577, disc\_loss: 1.2061  
Time for epoch 259 is 2.6246 sec  
Epoch 260, gen\_loss: 1.4220, disc\_loss: 1.0851  
Time for epoch 260 is 2.7252 sec  
Epoch 261, gen\_loss: 1.3585, disc\_loss: 1.1567  
Time for epoch 261 is 2.7612 sec  
Epoch 262, gen\_loss: 1.2247, disc\_loss: 1.3895  
Time for epoch 262 is 2.9515 sec  
Epoch 263, gen\_loss: 1.2732, disc\_loss: 1.3774  
Time for epoch 263 is 2.7841 sec  
Epoch 264, gen\_loss: 1.4416, disc\_loss: 1.3462  
Time for epoch 264 is 2.6572 sec  
Epoch 265, gen\_loss: 1.3250, disc\_loss: 1.3090  
Time for epoch 265 is 2.7527 sec  
Epoch 266, gen\_loss: 1.4844, disc\_loss: 1.2132  
Time for epoch 266 is 2.5791 sec  
Epoch 267, gen\_loss: 1.8190, disc\_loss: 0.9538  
Time for epoch 267 is 2.7806 sec  
Epoch 268, gen\_loss: 2.0903, disc\_loss: 0.7580  
Time for epoch 268 is 2.7876 sec  
Epoch 269, gen\_loss: 1.7547, disc\_loss: 0.9348  
Time for epoch 269 is 2.7605 sec  
Epoch 270, gen\_loss: 1.7218, disc\_loss: 0.9265

Time for epoch 270 is 2.8252 sec  
Epoch 271, gen\_loss: 1.8513, disc\_loss: 0.9348  
Time for epoch 271 is 2.7492 sec  
Epoch 272, gen\_loss: 1.8183, disc\_loss: 1.0588  
Time for epoch 272 is 2.6567 sec  
Epoch 273, gen\_loss: 1.6136, disc\_loss: 0.9468  
Time for epoch 273 is 2.7929 sec  
Epoch 274, gen\_loss: 1.4519, disc\_loss: 1.0622  
Time for epoch 274 is 2.7231 sec  
Epoch 275, gen\_loss: 1.4036, disc\_loss: 1.0936  
Time for epoch 275 is 2.8790 sec  
Epoch 276, gen\_loss: 1.5005, disc\_loss: 0.9928  
Time for epoch 276 is 2.8418 sec  
Epoch 277, gen\_loss: 1.5284, disc\_loss: 1.1343  
Time for epoch 277 is 2.9306 sec  
Epoch 278, gen\_loss: 1.3337, disc\_loss: 1.1566  
Time for epoch 278 is 2.7686 sec  
Epoch 279, gen\_loss: 1.3872, disc\_loss: 1.1414  
Time for epoch 279 is 2.7699 sec  
Epoch 280, gen\_loss: 1.5129, disc\_loss: 1.0755  
Time for epoch 280 is 2.8942 sec  
Epoch 281, gen\_loss: 1.1406, disc\_loss: 1.5051  
Time for epoch 281 is 2.6282 sec  
Epoch 282, gen\_loss: 1.4127, disc\_loss: 1.1997  
Time for epoch 282 is 2.6639 sec  
Epoch 283, gen\_loss: 1.6613, disc\_loss: 1.1306  
Time for epoch 283 is 2.7509 sec  
Epoch 284, gen\_loss: 1.4131, disc\_loss: 1.1062  
Time for epoch 284 is 2.8086 sec  
Epoch 285, gen\_loss: 1.6414, disc\_loss: 1.0623  
Time for epoch 285 is 2.9261 sec  
Epoch 286, gen\_loss: 1.4476, disc\_loss: 1.1518  
Time for epoch 286 is 2.8268 sec  
Epoch 287, gen\_loss: 1.3745, disc\_loss: 1.1608  
Time for epoch 287 is 2.8192 sec  
Epoch 288, gen\_loss: 1.3930, disc\_loss: 1.2188  
Time for epoch 288 is 2.7996 sec  
Epoch 289, gen\_loss: 1.6432, disc\_loss: 0.8799  
Time for epoch 289 is 2.7188 sec  
Epoch 290, gen\_loss: 1.5918, disc\_loss: 0.9818  
Time for epoch 290 is 2.7611 sec  
Epoch 291, gen\_loss: 1.8098, disc\_loss: 0.8682  
Time for epoch 291 is 2.8230 sec  
Epoch 292, gen\_loss: 1.5497, disc\_loss: 1.1377  
Time for epoch 292 is 2.6379 sec  
Epoch 293, gen\_loss: 1.4540, disc\_loss: 1.1749  
Time for epoch 293 is 2.8885 sec  
Epoch 294, gen\_loss: 1.3085, disc\_loss: 1.3871  
Time for epoch 294 is 2.7974 sec  
Epoch 295, gen\_loss: 1.6358, disc\_loss: 1.1094  
Time for epoch 295 is 2.8617 sec  
Epoch 296, gen\_loss: 1.6265, disc\_loss: 1.0766  
Time for epoch 296 is 2.7733 sec  
Epoch 297, gen\_loss: 1.4983, disc\_loss: 1.0836  
Time for epoch 297 is 2.8805 sec  
Epoch 298, gen\_loss: 1.6753, disc\_loss: 0.9909  
Time for epoch 298 is 2.6323 sec  
Epoch 299, gen\_loss: 1.9241, disc\_loss: 0.7800  
Time for epoch 299 is 2.9880 sec  
Epoch 300, gen\_loss: 1.2872, disc\_loss: 1.2479  
Time for epoch 300 is 2.9007 sec  
Epoch 301, gen\_loss: 1.6593, disc\_loss: 0.9086  
Time for epoch 301 is 2.8858 sec

Epoch 302, gen\_loss: 1.6631, disc\_loss: 0.9807  
Time for epoch 302 is 2.8047 sec  
Epoch 303, gen\_loss: 1.6169, disc\_loss: 0.9004  
Time for epoch 303 is 2.9854 sec  
Epoch 304, gen\_loss: 1.4279, disc\_loss: 1.1204  
Time for epoch 304 is 2.9919 sec  
Epoch 305, gen\_loss: 1.4570, disc\_loss: 1.2273  
Time for epoch 305 is 2.9441 sec  
Epoch 306, gen\_loss: 1.5632, disc\_loss: 1.1381  
Time for epoch 306 is 2.7382 sec  
Epoch 307, gen\_loss: 1.3286, disc\_loss: 1.2466  
Time for epoch 307 is 2.7870 sec  
Epoch 308, gen\_loss: 1.4674, disc\_loss: 1.2308  
Time for epoch 308 is 2.7111 sec  
Epoch 309, gen\_loss: 1.4283, disc\_loss: 1.2386  
Time for epoch 309 is 2.7855 sec  
Epoch 310, gen\_loss: 1.7795, disc\_loss: 0.8981  
Time for epoch 310 is 2.6546 sec  
Epoch 311, gen\_loss: 1.4665, disc\_loss: 1.0958  
Time for epoch 311 is 2.6369 sec  
Epoch 312, gen\_loss: 1.3966, disc\_loss: 1.1400  
Time for epoch 312 is 2.6846 sec  
Epoch 313, gen\_loss: 1.3346, disc\_loss: 1.3311  
Time for epoch 313 is 2.6534 sec  
Epoch 314, gen\_loss: 1.6244, disc\_loss: 1.0815  
Time for epoch 314 is 2.7793 sec  
Epoch 315, gen\_loss: 1.6190, disc\_loss: 1.0886  
Time for epoch 315 is 2.7612 sec  
Epoch 316, gen\_loss: 1.9007, disc\_loss: 0.9334  
Time for epoch 316 is 2.6074 sec  
Epoch 317, gen\_loss: 1.5427, disc\_loss: 1.0687  
Time for epoch 317 is 2.7645 sec  
Epoch 318, gen\_loss: 1.5660, disc\_loss: 1.0877  
Time for epoch 318 is 2.7915 sec  
Epoch 319, gen\_loss: 1.7837, disc\_loss: 1.0065  
Time for epoch 319 is 2.6891 sec  
Epoch 320, gen\_loss: 1.3288, disc\_loss: 1.2286  
Time for epoch 320 is 2.8767 sec  
Epoch 321, gen\_loss: 1.7182, disc\_loss: 0.8514  
Time for epoch 321 is 2.8580 sec  
Epoch 322, gen\_loss: 1.2799, disc\_loss: 1.2151  
Time for epoch 322 is 2.7880 sec  
Epoch 323, gen\_loss: 1.3362, disc\_loss: 1.2152  
Time for epoch 323 is 2.7481 sec  
Epoch 324, gen\_loss: 1.6702, disc\_loss: 0.9521  
Time for epoch 324 is 2.7856 sec  
Epoch 325, gen\_loss: 1.1675, disc\_loss: 1.3478  
Time for epoch 325 is 2.6974 sec  
Epoch 326, gen\_loss: 1.4348, disc\_loss: 1.2409  
Time for epoch 326 is 2.8468 sec  
Epoch 327, gen\_loss: 1.4106, disc\_loss: 1.3255  
Time for epoch 327 is 2.7237 sec  
Epoch 328, gen\_loss: 1.3221, disc\_loss: 1.5220  
Time for epoch 328 is 2.6975 sec  
Epoch 329, gen\_loss: 1.2438, disc\_loss: 1.5355  
Time for epoch 329 is 2.7704 sec  
Epoch 330, gen\_loss: 1.7441, disc\_loss: 1.1445  
Time for epoch 330 is 2.6853 sec  
Epoch 331, gen\_loss: 1.7485, disc\_loss: 0.9878  
Time for epoch 331 is 2.7803 sec  
Epoch 332, gen\_loss: 1.7707, disc\_loss: 0.9071  
Time for epoch 332 is 2.8351 sec  
Epoch 333, gen\_loss: 1.6581, disc\_loss: 0.9851

Time for epoch 333 is 2.7319 sec  
Epoch 334, gen\_loss: 1.7044, disc\_loss: 1.0054  
Time for epoch 334 is 2.7197 sec  
Epoch 335, gen\_loss: 1.5939, disc\_loss: 1.1581  
Time for epoch 335 is 2.6878 sec  
Epoch 336, gen\_loss: 1.7731, disc\_loss: 0.9612  
Time for epoch 336 is 2.7581 sec  
Epoch 337, gen\_loss: 1.7216, disc\_loss: 0.9264  
Time for epoch 337 is 2.7563 sec  
Epoch 338, gen\_loss: 1.5040, disc\_loss: 1.1345  
Time for epoch 338 is 2.6521 sec  
Epoch 339, gen\_loss: 1.6214, disc\_loss: 0.9291  
Time for epoch 339 is 2.6662 sec  
Epoch 340, gen\_loss: 1.7217, disc\_loss: 0.9582  
Time for epoch 340 is 2.7671 sec  
Epoch 341, gen\_loss: 1.5390, disc\_loss: 1.0010  
Time for epoch 341 is 2.7349 sec  
Epoch 342, gen\_loss: 1.3240, disc\_loss: 1.0776  
Time for epoch 342 is 2.7630 sec  
Epoch 343, gen\_loss: 1.3072, disc\_loss: 1.1885  
Time for epoch 343 is 2.9502 sec  
Epoch 344, gen\_loss: 1.3794, disc\_loss: 1.1979  
Time for epoch 344 is 2.7727 sec  
Epoch 345, gen\_loss: 1.4357, disc\_loss: 1.2087  
Time for epoch 345 is 2.8577 sec  
Epoch 346, gen\_loss: 1.3950, disc\_loss: 1.2965  
Time for epoch 346 is 2.9441 sec  
Epoch 347, gen\_loss: 1.1808, disc\_loss: 1.5667  
Time for epoch 347 is 2.7166 sec  
Epoch 348, gen\_loss: 1.5733, disc\_loss: 1.4118  
Time for epoch 348 is 2.8128 sec  
Epoch 349, gen\_loss: 1.4873, disc\_loss: 1.2795  
Time for epoch 349 is 2.7518 sec  
Epoch 350, gen\_loss: 1.4292, disc\_loss: 1.4488  
Time for epoch 350 is 2.9816 sec  
Epoch 351, gen\_loss: 1.6256, disc\_loss: 1.0396  
Time for epoch 351 is 2.8595 sec  
Epoch 352, gen\_loss: 1.8274, disc\_loss: 0.9704  
Time for epoch 352 is 2.9678 sec  
Epoch 353, gen\_loss: 1.4814, disc\_loss: 1.1546  
Time for epoch 353 is 2.7894 sec  
Epoch 354, gen\_loss: 1.6868, disc\_loss: 0.9467  
Time for epoch 354 is 2.9174 sec  
Epoch 355, gen\_loss: 1.7749, disc\_loss: 0.9340  
Time for epoch 355 is 2.8553 sec  
Epoch 356, gen\_loss: 1.4263, disc\_loss: 1.0692  
Time for epoch 356 is 2.7612 sec  
Epoch 357, gen\_loss: 1.4434, disc\_loss: 1.2290  
Time for epoch 357 is 2.8467 sec  
Epoch 358, gen\_loss: 1.6983, disc\_loss: 1.0236  
Time for epoch 358 is 2.7407 sec  
Epoch 359, gen\_loss: 1.6323, disc\_loss: 1.1382  
Time for epoch 359 is 2.6560 sec  
Epoch 360, gen\_loss: 1.2418, disc\_loss: 1.4237  
Time for epoch 360 is 2.7045 sec  
Epoch 361, gen\_loss: 1.9009, disc\_loss: 0.8905  
Time for epoch 361 is 2.8315 sec  
Epoch 362, gen\_loss: 1.4465, disc\_loss: 1.2576  
Time for epoch 362 is 2.7036 sec  
Epoch 363, gen\_loss: 1.6023, disc\_loss: 1.0113  
Time for epoch 363 is 2.7296 sec  
Epoch 364, gen\_loss: 1.6522, disc\_loss: 1.0643  
Time for epoch 364 is 2.7128 sec



Epoch 365, gen\_loss: 1.4827, disc\_loss: 1.1957  
Time for epoch 365 is 2.6435 sec  
Epoch 366, gen\_loss: 1.3175, disc\_loss: 1.2713  
Time for epoch 366 is 2.7248 sec  
Epoch 367, gen\_loss: 1.5251, disc\_loss: 1.1212  
Time for epoch 367 is 2.8105 sec  
Epoch 368, gen\_loss: 1.4801, disc\_loss: 1.1093  
Time for epoch 368 is 2.6388 sec  
Epoch 369, gen\_loss: 1.1733, disc\_loss: 1.4442  
Time for epoch 369 is 2.6636 sec  
Epoch 370, gen\_loss: 1.4051, disc\_loss: 1.3949  
Time for epoch 370 is 2.6614 sec  
Epoch 371, gen\_loss: 1.2759, disc\_loss: 1.4257  
Time for epoch 371 is 2.7569 sec  
Epoch 372, gen\_loss: 1.2629, disc\_loss: 1.4058  
Time for epoch 372 is 2.7644 sec  
Epoch 373, gen\_loss: 1.4763, disc\_loss: 1.1981  
Time for epoch 373 is 2.7033 sec  
Epoch 374, gen\_loss: 1.7870, disc\_loss: 1.0129  
Time for epoch 374 is 2.6969 sec  
Epoch 375, gen\_loss: 1.6267, disc\_loss: 1.1171  
Time for epoch 375 is 2.7625 sec  
Epoch 376, gen\_loss: 1.6906, disc\_loss: 0.9901  
Time for epoch 376 is 2.5614 sec  
Epoch 377, gen\_loss: 2.1171, disc\_loss: 0.7844  
Time for epoch 377 is 2.6701 sec  
Epoch 378, gen\_loss: 1.6431, disc\_loss: 1.0625  
Time for epoch 378 is 2.7074 sec  
Epoch 379, gen\_loss: 1.8434, disc\_loss: 0.9448  
Time for epoch 379 is 2.7283 sec  
Epoch 380, gen\_loss: 1.7540, disc\_loss: 1.0864  
Time for epoch 380 is 2.6521 sec  
Epoch 381, gen\_loss: 1.7315, disc\_loss: 0.9845  
Time for epoch 381 is 2.6950 sec  
Epoch 382, gen\_loss: 1.8021, disc\_loss: 0.8711  
Time for epoch 382 is 2.9052 sec  
Epoch 383, gen\_loss: 1.3917, disc\_loss: 1.0544  
Time for epoch 383 is 2.7707 sec  
Epoch 384, gen\_loss: 1.2276, disc\_loss: 1.3515  
Time for epoch 384 is 2.7151 sec  
Epoch 385, gen\_loss: 1.2708, disc\_loss: 1.2914  
Time for epoch 385 is 2.7132 sec  
Epoch 386, gen\_loss: 1.5658, disc\_loss: 1.0703  
Time for epoch 386 is 2.7494 sec  
Epoch 387, gen\_loss: 1.3154, disc\_loss: 1.2437  
Time for epoch 387 is 3.0056 sec  
Epoch 388, gen\_loss: 1.5441, disc\_loss: 1.0274  
Time for epoch 388 is 2.5559 sec  
Epoch 389, gen\_loss: 1.6212, disc\_loss: 0.9835  
Time for epoch 389 is 2.6446 sec  
Epoch 390, gen\_loss: 1.2129, disc\_loss: 1.3541  
Time for epoch 390 is 2.5817 sec  
Epoch 391, gen\_loss: 1.3833, disc\_loss: 1.2118  
Time for epoch 391 is 2.8167 sec  
Epoch 392, gen\_loss: 1.1538, disc\_loss: 1.5138  
Time for epoch 392 is 2.7704 sec  
Epoch 393, gen\_loss: 1.2686, disc\_loss: 1.6675  
Time for epoch 393 is 2.9392 sec  
Epoch 394, gen\_loss: 1.5654, disc\_loss: 1.1984  
Time for epoch 394 is 2.7612 sec  
Epoch 395, gen\_loss: 1.8232, disc\_loss: 0.9601  
Time for epoch 395 is 2.7999 sec  
Epoch 396, gen\_loss: 1.8364, disc\_loss: 0.8544

Time for epoch 396 is 2.7923 sec  
Epoch 397, gen\_loss: 1.7980, disc\_loss: 0.9042  
Time for epoch 397 is 2.8229 sec  
Epoch 398, gen\_loss: 1.3374, disc\_loss: 1.1641  
Time for epoch 398 is 2.7677 sec  
Epoch 399, gen\_loss: 1.8681, disc\_loss: 0.9643  
Time for epoch 399 is 2.7539 sec  
Epoch 400, gen\_loss: 1.5787, disc\_loss: 0.9883  
Time for epoch 400 is 2.8390 sec  
Epoch 401, gen\_loss: 1.6181, disc\_loss: 1.0654  
Time for epoch 401 is 2.6751 sec  
Epoch 402, gen\_loss: 1.3172, disc\_loss: 1.3174  
Time for epoch 402 is 2.7015 sec  
Epoch 403, gen\_loss: 1.7209, disc\_loss: 0.9637  
Time for epoch 403 is 2.7929 sec  
Epoch 404, gen\_loss: 1.2229, disc\_loss: 1.3733  
Time for epoch 404 is 2.7659 sec  
Epoch 405, gen\_loss: 1.3866, disc\_loss: 1.2168  
Time for epoch 405 is 2.7028 sec  
Epoch 406, gen\_loss: 1.2791, disc\_loss: 1.3222  
Time for epoch 406 is 2.7254 sec  
Epoch 407, gen\_loss: 1.3150, disc\_loss: 1.3072  
Time for epoch 407 is 2.7362 sec  
Epoch 408, gen\_loss: 1.2204, disc\_loss: 1.4637  
Time for epoch 408 is 2.7454 sec  
Epoch 409, gen\_loss: 2.0186, disc\_loss: 0.8106  
Time for epoch 409 is 2.7442 sec  
Epoch 410, gen\_loss: 1.6033, disc\_loss: 1.0769  
Time for epoch 410 is 2.7539 sec  
Epoch 411, gen\_loss: 1.4859, disc\_loss: 1.2395  
Time for epoch 411 is 2.8115 sec  
Epoch 412, gen\_loss: 1.4391, disc\_loss: 1.1866  
Time for epoch 412 is 2.7873 sec  
Epoch 413, gen\_loss: 1.5675, disc\_loss: 1.0501  
Time for epoch 413 is 2.7097 sec  
Epoch 414, gen\_loss: 1.4487, disc\_loss: 1.2659  
Time for epoch 414 is 2.8515 sec  
Epoch 415, gen\_loss: 1.2983, disc\_loss: 1.3585  
Time for epoch 415 is 2.6504 sec  
Epoch 416, gen\_loss: 1.3784, disc\_loss: 1.3412  
Time for epoch 416 is 2.6366 sec  
Epoch 417, gen\_loss: 1.5210, disc\_loss: 1.1293  
Time for epoch 417 is 2.6563 sec  
Epoch 418, gen\_loss: 1.6260, disc\_loss: 1.0689  
Time for epoch 418 is 2.6230 sec  
Epoch 419, gen\_loss: 1.6314, disc\_loss: 1.1181  
Time for epoch 419 is 2.7198 sec  
Epoch 420, gen\_loss: 1.5543, disc\_loss: 1.1009  
Time for epoch 420 is 2.7455 sec  
Epoch 421, gen\_loss: 1.3679, disc\_loss: 1.2846  
Time for epoch 421 is 2.6184 sec  
Epoch 422, gen\_loss: 1.9187, disc\_loss: 1.0367  
Time for epoch 422 is 2.6684 sec  
Epoch 423, gen\_loss: 1.5627, disc\_loss: 1.1382  
Time for epoch 423 is 2.6449 sec  
Epoch 424, gen\_loss: 1.6289, disc\_loss: 1.1250  
Time for epoch 424 is 2.6629 sec  
Epoch 425, gen\_loss: 2.0070, disc\_loss: 0.8487  
Time for epoch 425 is 2.6019 sec  
Epoch 426, gen\_loss: 1.5220, disc\_loss: 1.1775  
Time for epoch 426 is 2.6972 sec  
Epoch 427, gen\_loss: 1.7632, disc\_loss: 0.8631  
Time for epoch 427 is 2.6447 sec

Epoch 428, gen\_loss: 1.5201, disc\_loss: 1.0043  
Time for epoch 428 is 2.8476 sec  
Epoch 429, gen\_loss: 1.4938, disc\_loss: 1.0774  
Time for epoch 429 is 2.6557 sec  
Epoch 430, gen\_loss: 1.3933, disc\_loss: 1.2085  
Time for epoch 430 is 2.7787 sec  
Epoch 431, gen\_loss: 1.2867, disc\_loss: 1.2714  
Time for epoch 431 is 2.7234 sec  
Epoch 432, gen\_loss: 1.4389, disc\_loss: 1.2660  
Time for epoch 432 is 2.8022 sec  
Epoch 433, gen\_loss: 1.5362, disc\_loss: 1.0666  
Time for epoch 433 is 2.8580 sec  
Epoch 434, gen\_loss: 1.2960, disc\_loss: 1.3087  
Time for epoch 434 is 2.9488 sec  
Epoch 435, gen\_loss: 1.4256, disc\_loss: 1.0963  
Time for epoch 435 is 2.8921 sec  
Epoch 436, gen\_loss: 1.5434, disc\_loss: 0.9924  
Time for epoch 436 is 2.8644 sec  
Epoch 437, gen\_loss: 1.2536, disc\_loss: 1.3110  
Time for epoch 437 is 2.7915 sec  
Epoch 438, gen\_loss: 1.0823, disc\_loss: 1.5584  
Time for epoch 438 is 2.9781 sec  
Epoch 439, gen\_loss: 1.3536, disc\_loss: 1.2896  
Time for epoch 439 is 2.6991 sec  
Epoch 440, gen\_loss: 1.6026, disc\_loss: 1.2936  
Time for epoch 440 is 2.7059 sec  
Epoch 441, gen\_loss: 1.5154, disc\_loss: 1.1366  
Time for epoch 441 is 2.7734 sec  
Epoch 442, gen\_loss: 1.5212, disc\_loss: 1.0960  
Time for epoch 442 is 2.7291 sec  
Epoch 443, gen\_loss: 1.3036, disc\_loss: 1.3351  
Time for epoch 443 is 2.5449 sec  
Epoch 444, gen\_loss: 1.4602, disc\_loss: 1.4262  
Time for epoch 444 is 2.7202 sec  
Epoch 445, gen\_loss: 1.5130, disc\_loss: 1.3583  
Time for epoch 445 is 2.8305 sec  
Epoch 446, gen\_loss: 1.4605, disc\_loss: 1.3482  
Time for epoch 446 is 2.6786 sec  
Epoch 447, gen\_loss: 1.6412, disc\_loss: 1.0524  
Time for epoch 447 is 2.7903 sec  
Epoch 448, gen\_loss: 1.8714, disc\_loss: 0.8906  
Time for epoch 448 is 2.8496 sec  
Epoch 449, gen\_loss: 1.8497, disc\_loss: 0.9716  
Time for epoch 449 is 2.9732 sec  
Epoch 450, gen\_loss: 1.7351, disc\_loss: 0.9426  
Time for epoch 450 is 2.8633 sec  
Epoch 451, gen\_loss: 1.5395, disc\_loss: 1.1873  
Time for epoch 451 is 2.7796 sec  
Epoch 452, gen\_loss: 1.6101, disc\_loss: 1.0879  
Time for epoch 452 is 2.7569 sec  
Epoch 453, gen\_loss: 1.7661, disc\_loss: 1.0346  
Time for epoch 453 is 2.8468 sec  
Epoch 454, gen\_loss: 1.4266, disc\_loss: 1.1831  
Time for epoch 454 is 2.5675 sec  
Epoch 455, gen\_loss: 1.6030, disc\_loss: 1.1439  
Time for epoch 455 is 2.8151 sec  
Epoch 456, gen\_loss: 1.4727, disc\_loss: 1.0945  
Time for epoch 456 is 2.7031 sec  
Epoch 457, gen\_loss: 1.7364, disc\_loss: 0.9952  
Time for epoch 457 is 2.7427 sec  
Epoch 458, gen\_loss: 1.7599, disc\_loss: 0.9355  
Time for epoch 458 is 2.6672 sec  
Epoch 459, gen\_loss: 1.4262, disc\_loss: 1.3085

Time for epoch 459 is 2.8015 sec  
Epoch 460, gen\_loss: 1.5590, disc\_loss: 1.1239  
Time for epoch 460 is 2.6729 sec  
Epoch 461, gen\_loss: 1.5915, disc\_loss: 1.1408  
Time for epoch 461 is 2.6674 sec  
Epoch 462, gen\_loss: 1.5240, disc\_loss: 1.3077  
Time for epoch 462 is 2.8612 sec  
Epoch 463, gen\_loss: 2.2941, disc\_loss: 0.7807  
Time for epoch 463 is 2.8878 sec  
Epoch 464, gen\_loss: 1.4038, disc\_loss: 1.2953  
Time for epoch 464 is 2.6226 sec  
Epoch 465, gen\_loss: 1.5378, disc\_loss: 1.0863  
Time for epoch 465 is 2.7499 sec  
Epoch 466, gen\_loss: 1.9171, disc\_loss: 0.8427  
Time for epoch 466 is 2.6729 sec  
Epoch 467, gen\_loss: 1.5662, disc\_loss: 1.0975  
Time for epoch 467 is 2.6927 sec  
Epoch 468, gen\_loss: 1.2825, disc\_loss: 1.3178  
Time for epoch 468 is 2.6455 sec  
Epoch 469, gen\_loss: 1.4879, disc\_loss: 1.1401  
Time for epoch 469 is 2.6707 sec  
Epoch 470, gen\_loss: 1.4572, disc\_loss: 1.1605  
Time for epoch 470 is 2.7070 sec  
Epoch 471, gen\_loss: 1.5924, disc\_loss: 1.0746  
Time for epoch 471 is 2.7430 sec  
Epoch 472, gen\_loss: 1.5058, disc\_loss: 1.0792  
Time for epoch 472 is 2.8028 sec  
Epoch 473, gen\_loss: 1.2779, disc\_loss: 1.1645  
Time for epoch 473 is 2.7782 sec  
Epoch 474, gen\_loss: 1.4913, disc\_loss: 1.1427  
Time for epoch 474 is 2.7103 sec  
Epoch 475, gen\_loss: 1.3261, disc\_loss: 1.3336  
Time for epoch 475 is 2.6980 sec  
Epoch 476, gen\_loss: 1.0999, disc\_loss: 1.6573  
Time for epoch 476 is 2.7192 sec  
Epoch 477, gen\_loss: 1.4050, disc\_loss: 1.3802  
Time for epoch 477 is 2.7733 sec  
Epoch 478, gen\_loss: 1.2912, disc\_loss: 1.4844  
Time for epoch 478 is 2.8197 sec  
Epoch 479, gen\_loss: 1.6490, disc\_loss: 1.1076  
Time for epoch 479 is 2.8114 sec  
Epoch 480, gen\_loss: 1.4154, disc\_loss: 1.2301  
Time for epoch 480 is 2.6944 sec  
Epoch 481, gen\_loss: 1.5465, disc\_loss: 1.1103  
Time for epoch 481 is 2.6894 sec  
Epoch 482, gen\_loss: 1.3327, disc\_loss: 1.2531  
Time for epoch 482 is 2.6376 sec  
Epoch 483, gen\_loss: 1.4315, disc\_loss: 1.1828  
Time for epoch 483 is 2.6310 sec  
Epoch 484, gen\_loss: 1.6578, disc\_loss: 1.1946  
Time for epoch 484 is 2.6974 sec  
Epoch 485, gen\_loss: 1.5816, disc\_loss: 1.2462  
Time for epoch 485 is 2.6284 sec  
Epoch 486, gen\_loss: 1.5966, disc\_loss: 1.0815  
Time for epoch 486 is 2.8203 sec  
Epoch 487, gen\_loss: 1.6828, disc\_loss: 1.0564  
Time for epoch 487 is 2.7773 sec  
Epoch 488, gen\_loss: 1.7555, disc\_loss: 1.0164  
Time for epoch 488 is 2.8441 sec  
Epoch 489, gen\_loss: 1.9055, disc\_loss: 0.8449  
Time for epoch 489 is 2.9597 sec  
Epoch 490, gen\_loss: 1.4928, disc\_loss: 1.0706  
Time for epoch 490 is 2.8854 sec

Epoch 491, gen\_loss: 1.2476, disc\_loss: 1.2289  
Time for epoch 491 is 2.8749 sec  
Epoch 492, gen\_loss: 1.3137, disc\_loss: 1.2651  
Time for epoch 492 is 3.0401 sec  
Epoch 493, gen\_loss: 1.3372, disc\_loss: 1.2935  
Time for epoch 493 is 2.7728 sec  
Epoch 494, gen\_loss: 1.2992, disc\_loss: 1.3009  
Time for epoch 494 is 2.7506 sec  
Epoch 495, gen\_loss: 1.2431, disc\_loss: 1.4563  
Time for epoch 495 is 2.7109 sec  
Epoch 496, gen\_loss: 1.4048, disc\_loss: 1.2899  
Time for epoch 496 is 2.8585 sec  
Epoch 497, gen\_loss: 1.5306, disc\_loss: 1.2232  
Time for epoch 497 is 2.8410 sec  
Epoch 498, gen\_loss: 1.7924, disc\_loss: 0.9876  
Time for epoch 498 is 2.9499 sec  
Epoch 499, gen\_loss: 1.6207, disc\_loss: 1.1187  
Time for epoch 499 is 2.6812 sec  
Epoch 500, gen\_loss: 1.4136, disc\_loss: 1.2424  
Time for epoch 500 is 2.8371 sec  
Epoch 501, gen\_loss: 2.1174, disc\_loss: 0.7992  
Time for epoch 501 is 2.7750 sec  
Epoch 502, gen\_loss: 1.4939, disc\_loss: 1.1240  
Time for epoch 502 is 2.8081 sec  
Epoch 503, gen\_loss: 1.2996, disc\_loss: 1.2645  
Time for epoch 503 is 2.6079 sec  
Epoch 504, gen\_loss: 1.7164, disc\_loss: 1.1071  
Time for epoch 504 is 2.7863 sec  
Epoch 505, gen\_loss: 1.6398, disc\_loss: 1.0651  
Time for epoch 505 is 2.8250 sec  
Epoch 506, gen\_loss: 1.5029, disc\_loss: 1.1538  
Time for epoch 506 is 2.9609 sec  
Epoch 507, gen\_loss: 1.7585, disc\_loss: 1.0341  
Time for epoch 507 is 2.8943 sec  
Epoch 508, gen\_loss: 1.9368, disc\_loss: 0.8680  
Time for epoch 508 is 2.8941 sec  
Epoch 509, gen\_loss: 1.4061, disc\_loss: 1.1411  
Time for epoch 509 is 2.8259 sec  
Epoch 510, gen\_loss: 1.5924, disc\_loss: 1.0409  
Time for epoch 510 is 2.6853 sec  
Epoch 511, gen\_loss: 1.6224, disc\_loss: 0.9173  
Time for epoch 511 is 2.7522 sec  
Epoch 512, gen\_loss: 1.4257, disc\_loss: 1.0282  
Time for epoch 512 is 2.7143 sec  
Epoch 513, gen\_loss: 1.3125, disc\_loss: 1.1940  
Time for epoch 513 is 2.5786 sec  
Epoch 514, gen\_loss: 1.3834, disc\_loss: 1.2133  
Time for epoch 514 is 2.7398 sec  
Epoch 515, gen\_loss: 1.1962, disc\_loss: 1.5243  
Time for epoch 515 is 2.8060 sec  
Epoch 516, gen\_loss: 1.2929, disc\_loss: 1.4014  
Time for epoch 516 is 2.9010 sec  
Epoch 517, gen\_loss: 1.4878, disc\_loss: 1.2858  
Time for epoch 517 is 2.8596 sec  
Epoch 518, gen\_loss: 1.1814, disc\_loss: 1.6164  
Time for epoch 518 is 2.7642 sec  
Epoch 519, gen\_loss: 1.4546, disc\_loss: 1.2266  
Time for epoch 519 is 2.6708 sec  
Epoch 520, gen\_loss: 1.6875, disc\_loss: 1.0968  
Time for epoch 520 is 2.7628 sec  
Epoch 521, gen\_loss: 1.3046, disc\_loss: 1.2740  
Time for epoch 521 is 2.6423 sec  
Epoch 522, gen\_loss: 1.4252, disc\_loss: 1.2737

Time for epoch 522 is 2.7360 sec  
Epoch 523, gen\_loss: 1.4886, disc\_loss: 1.2958  
Time for epoch 523 is 2.7144 sec  
Epoch 524, gen\_loss: 1.6100, disc\_loss: 1.1328  
Time for epoch 524 is 2.7411 sec  
Epoch 525, gen\_loss: 1.5873, disc\_loss: 1.1115  
Time for epoch 525 is 2.7918 sec  
Epoch 526, gen\_loss: 1.6348, disc\_loss: 0.9571  
Time for epoch 526 is 2.7395 sec  
Epoch 527, gen\_loss: 1.7451, disc\_loss: 0.9149  
Time for epoch 527 is 2.7273 sec  
Epoch 528, gen\_loss: 1.7042, disc\_loss: 1.0534  
Time for epoch 528 is 2.7114 sec  
Epoch 529, gen\_loss: 1.5268, disc\_loss: 1.1351  
Time for epoch 529 is 2.6077 sec  
Epoch 530, gen\_loss: 1.4707, disc\_loss: 1.1536  
Time for epoch 530 is 2.7618 sec  
Epoch 531, gen\_loss: 1.7358, disc\_loss: 0.9980  
Time for epoch 531 is 2.7217 sec  
Epoch 532, gen\_loss: 1.6163, disc\_loss: 1.0855  
Time for epoch 532 is 2.6190 sec  
Epoch 533, gen\_loss: 1.5931, disc\_loss: 1.1077  
Time for epoch 533 is 2.6307 sec  
Epoch 534, gen\_loss: 1.3834, disc\_loss: 1.2589  
Time for epoch 534 is 2.6582 sec  
Epoch 535, gen\_loss: 1.6760, disc\_loss: 1.0114  
Time for epoch 535 is 2.5883 sec  
Epoch 536, gen\_loss: 1.5303, disc\_loss: 1.1686  
Time for epoch 536 is 2.8013 sec  
Epoch 537, gen\_loss: 1.3518, disc\_loss: 1.3193  
Time for epoch 537 is 2.6143 sec  
Epoch 538, gen\_loss: 1.5777, disc\_loss: 1.1940  
Time for epoch 538 is 2.6405 sec  
Epoch 539, gen\_loss: 1.3867, disc\_loss: 1.5501  
Time for epoch 539 is 2.5839 sec  
Epoch 540, gen\_loss: 1.5092, disc\_loss: 1.3023  
Time for epoch 540 is 2.8567 sec  
Epoch 541, gen\_loss: 1.4207, disc\_loss: 1.3526  
Time for epoch 541 is 2.8238 sec  
Epoch 542, gen\_loss: 1.6522, disc\_loss: 1.1371  
Time for epoch 542 is 2.6856 sec  
Epoch 543, gen\_loss: 1.8273, disc\_loss: 0.9875  
Time for epoch 543 is 2.7811 sec  
Epoch 544, gen\_loss: 1.4701, disc\_loss: 1.1248  
Time for epoch 544 is 2.7667 sec  
Epoch 545, gen\_loss: 1.7753, disc\_loss: 1.0201  
Time for epoch 545 is 2.9215 sec  
Epoch 546, gen\_loss: 1.5912, disc\_loss: 1.1389  
Time for epoch 546 is 2.8773 sec  
Epoch 547, gen\_loss: 1.3136, disc\_loss: 1.3174  
Time for epoch 547 is 2.7584 sec  
Epoch 548, gen\_loss: 1.9351, disc\_loss: 0.9570  
Time for epoch 548 is 2.8885 sec  
Epoch 549, gen\_loss: 1.5576, disc\_loss: 1.1265  
Time for epoch 549 is 2.8406 sec  
Epoch 550, gen\_loss: 1.4023, disc\_loss: 1.1809  
Time for epoch 550 is 2.8195 sec  
Epoch 551, gen\_loss: 1.5920, disc\_loss: 1.1071  
Time for epoch 551 is 2.6180 sec  
Epoch 552, gen\_loss: 1.6358, disc\_loss: 1.1785  
Time for epoch 552 is 2.6449 sec  
Epoch 553, gen\_loss: 1.3750, disc\_loss: 1.1346  
Time for epoch 553 is 2.6846 sec

Epoch 554, gen\_loss: 1.3347, disc\_loss: 1.2719  
Time for epoch 554 is 2.6199 sec  
Epoch 555, gen\_loss: 1.4284, disc\_loss: 1.2857  
Time for epoch 555 is 2.7669 sec  
Epoch 556, gen\_loss: 1.5654, disc\_loss: 1.1866  
Time for epoch 556 is 2.5750 sec  
Epoch 557, gen\_loss: 1.8400, disc\_loss: 1.0466  
Time for epoch 557 is 2.7649 sec  
Epoch 558, gen\_loss: 1.5340, disc\_loss: 1.1476  
Time for epoch 558 is 2.6049 sec  
Epoch 559, gen\_loss: 1.4575, disc\_loss: 1.1819  
Time for epoch 559 is 2.6871 sec  
Epoch 560, gen\_loss: 1.5891, disc\_loss: 1.0939  
Time for epoch 560 is 2.7941 sec  
Epoch 561, gen\_loss: 1.5378, disc\_loss: 1.0439  
Time for epoch 561 is 2.6607 sec  
Epoch 562, gen\_loss: 1.6763, disc\_loss: 1.0366  
Time for epoch 562 is 2.7165 sec  
Epoch 563, gen\_loss: 1.2656, disc\_loss: 1.3536  
Time for epoch 563 is 2.6650 sec  
Epoch 564, gen\_loss: 1.4976, disc\_loss: 1.0266  
Time for epoch 564 is 2.7758 sec  
Epoch 565, gen\_loss: 1.5531, disc\_loss: 1.1300  
Time for epoch 565 is 2.6458 sec  
Epoch 566, gen\_loss: 1.2524, disc\_loss: 1.5051  
Time for epoch 566 is 2.7671 sec  
Epoch 567, gen\_loss: 1.5045, disc\_loss: 1.3402  
Time for epoch 567 is 2.8564 sec  
Epoch 568, gen\_loss: 1.7216, disc\_loss: 1.2275  
Time for epoch 568 is 2.7879 sec  
Epoch 569, gen\_loss: 1.1315, disc\_loss: 1.4727  
Time for epoch 569 is 2.6206 sec  
Epoch 570, gen\_loss: 1.2762, disc\_loss: 1.4192  
Time for epoch 570 is 2.7381 sec  
Epoch 571, gen\_loss: 1.8761, disc\_loss: 1.0647  
Time for epoch 571 is 2.7208 sec  
Epoch 572, gen\_loss: 1.5518, disc\_loss: 1.1672  
Time for epoch 572 is 2.7640 sec  
Epoch 573, gen\_loss: 1.5545, disc\_loss: 1.0629  
Time for epoch 573 is 2.8095 sec  
Epoch 574, gen\_loss: 1.4991, disc\_loss: 1.1363  
Time for epoch 574 is 2.8818 sec  
Epoch 575, gen\_loss: 1.8073, disc\_loss: 0.9132  
Time for epoch 575 is 2.6395 sec  
Epoch 576, gen\_loss: 1.9095, disc\_loss: 0.8053  
Time for epoch 576 is 2.8770 sec  
Epoch 577, gen\_loss: 1.6568, disc\_loss: 0.8860  
Time for epoch 577 is 2.7472 sec  
Epoch 578, gen\_loss: 1.1618, disc\_loss: 1.3050  
Time for epoch 578 is 2.6350 sec  
Epoch 579, gen\_loss: 1.5038, disc\_loss: 1.0389  
Time for epoch 579 is 2.6233 sec  
Epoch 580, gen\_loss: 1.0898, disc\_loss: 1.5164  
Time for epoch 580 is 2.9061 sec  
Epoch 581, gen\_loss: 1.2877, disc\_loss: 1.3794  
Time for epoch 581 is 2.7541 sec  
Epoch 582, gen\_loss: 1.1556, disc\_loss: 1.4464  
Time for epoch 582 is 2.8219 sec  
Epoch 583, gen\_loss: 1.5210, disc\_loss: 1.2693  
Time for epoch 583 is 2.8192 sec  
Epoch 584, gen\_loss: 1.1246, disc\_loss: 1.5835  
Time for epoch 584 is 2.7571 sec  
Epoch 585, gen\_loss: 1.5627, disc\_loss: 1.2289



```

Time for epoch 585 is 2.8443 sec
Epoch 586, gen_loss: 1.2672, disc_loss: 1.5386
Time for epoch 586 is 2.7702 sec
Epoch 587, gen_loss: 1.4507, disc_loss: 1.3906
Time for epoch 587 is 2.7806 sec
Epoch 588, gen_loss: 1.7697, disc_loss: 1.1264
Time for epoch 588 is 2.7343 sec
Epoch 589, gen_loss: 1.4761, disc_loss: 1.2017
Time for epoch 589 is 2.6996 sec
Epoch 590, gen_loss: 1.7790, disc_loss: 0.9650
Time for epoch 590 is 2.7334 sec
Epoch 591, gen_loss: 1.2539, disc_loss: 1.3430
Time for epoch 591 is 2.7755 sec
Epoch 592, gen_loss: 1.6775, disc_loss: 0.9512
Time for epoch 592 is 2.8040 sec
Epoch 593, gen_loss: 1.8180, disc_loss: 0.8583
Time for epoch 593 is 2.7300 sec
Epoch 594, gen_loss: 1.6375, disc_loss: 0.9610
Time for epoch 594 is 2.7661 sec
Epoch 595, gen_loss: 1.6283, disc_loss: 0.9444
Time for epoch 595 is 2.7414 sec
Epoch 596, gen_loss: 1.3661, disc_loss: 1.1386
Time for epoch 596 is 2.6704 sec
Epoch 597, gen_loss: 1.3053, disc_loss: 1.2511
Time for epoch 597 is 2.7331 sec
Epoch 598, gen_loss: 1.5153, disc_loss: 1.2351
Time for epoch 598 is 2.8413 sec
Epoch 599, gen_loss: 1.6095, disc_loss: 1.1348
Time for epoch 599 is 2.7835 sec
Epoch 600, gen_loss: 1.2252, disc_loss: 1.5143
Time for epoch 600 is 2.7972 sec

```

## Evaluation

`dataset/testData.pkl` is a pandas dataframe containing testing text with attributes 'ID' and 'Captions'.

- 'ID': text ID used to name generated image.
- 'Captions': text used as condition to generate image.

For each captions, you need to generate **inference\_ID.png** to evaluate quality of generated image. You must name the generated image in this format, otherwise we cannot evaluate your images.

## Testing Dataset

If you change anything during preprocessing of training dataset, you must make sure same operations have be done in testing dataset.

```

In [26]: def testing_data_generator(caption, index):
          caption = tf.cast(caption, tf.float32)
          return caption, index

def testing_dataset_generator(batch_size, data_generator):
    data = pd.read_pickle('./dataset/testData.pkl')
    captions = data['Captions'].values
    caption = []

```

```

for i in range(len(captions)):
    caption.append(captions[i])
caption = np.asarray(caption)
caption = caption.astype(np.int)
index = data['ID'].values
index = np.asarray(index)

dataset = tf.data.Dataset.from_tensor_slices((caption, index))
dataset = dataset.map(data_generator, num_parallel_calls=tf.data.experimental.AUTOTUNE)
dataset = dataset.repeat().batch(batch_size)

return dataset

```

```
In [27]: testing_dataset = testing_dataset_generator(hparas['BATCH_SIZE'], testing_data_generator)
```

```

/tmp/ipykernel_29626/466323065.py:12: DeprecationWarning: `np.int` is a deprecated alias
for the builtin `int`. To silence this warning, use `int` by itself. Doing this will not
modify any behavior and is safe. When replacing `np.int`, you may wish to use e.g. `np.i
nt64` or `np.int32` to specify the precision. If you wish to review your current use, ch
eck the release note link for additional information.
Deprecated in NumPy 1.20; for more details and guidance: https://numpy.org/devdocs/relea
se/1.20.0-notes.html#deprecations
caption = caption.astype(np.int)

```

```
In [28]: data = pd.read_pickle('./dataset/testData.pkl')
captions = data['Captions'].values

NUM_TEST = len(captions)
EPOCH_TEST = int(NUM_TEST / hparas['BATCH_SIZE'])
```

## Inference

```
In [32]: if not os.path.exists('./inference/demo'):
os.makedirs('./inference/demo')
```

```
In [33]: def inference(dataset):
hidden = text_encoder.initialize_hidden_state()
sample_size = hparas['BATCH_SIZE']
sample_seed = np.random.normal(loc=0.0, scale=1.0, size=(sample_size, hparas['Z_DIM']

step = 0
start = time.time()
for captions, idx in dataset:
    if step > EPOCH_TEST:
        break

    fake_image = test_step(captions, sample_seed, hidden)
    step += 1
    for i in range(hparas['BATCH_SIZE']):
        plt.imsave('./inference/demo/inference_{:04d}.jpg'.format(idx[i]), fake_imag

print('Time for inference is {:.4f} sec'.format(time.time()-start))

```

```
In [34]: checkpoint.restore(checkpoint_dir + '/ckpt-1')
```

```
Out[34]: <tensorflow.python.checkpoint.checkpoint.CheckpointLoadStatus at 0x7f640c349430>
```

```
In [35]: inference(testing_dataset)
```

```
Time for inference is 1.5614 sec
```

# Inception Score & Cosine Similarity

In this competition, we use both [inception score](#) and cosine distance as our final score to evaluate quality and diversity of generated images. You can find more details about inception score in [this article](#). The final score is based on:

- Similarity of images and the given contents. How similar are the generated images and the given texts?
- KL divergence of generated images. Are the generated images very diverse?

After generating images with given testing texts, you have to run evaluation script to generate `score.csv` file, and then upload it to Kaggle to get the final score.

## 1. Run evaluation script to generate score.csv

1. Open terminal and move to the folder containing `inception_score.py`. Otherwise you have to modify the path used in the file.
2. Run `python ./inception_score.py [argv1] [argv2] [argv3]`
  - argv1: directory of generated image (inference).
  - argv2: directory of output file and its name.
  - argv3: batch size. Please set batch size to 1, 2, 3, 7, 9, 21, 39 to avoid remainder.

For example, run following command `python inception_score.py ../inference/demo ../score_demo.csv 39`.

It is better for you to know that evaluation needs to run on GPUs, please make sure the GPU resource is available.

## 2. Submit

Submit `score.csv` to [DataLabCup: Reverse Image Caption](#)

# Precautions

## Competition timeline

- 2023/11/30(Thur) competition announced.
- 2023/12/22(Thur) 08:00(TW) competition deadline.
- 2023/12/24(Sun) 23:59(TW) report deadline.
- 2023/12/28(Thur) winner team share.

## Scoring

- Ranking of **private** leaderboard of competition. **(50%)**
  - team name should be exactly matched with [google sheet](#) or you will get zero in this part

- Inference images. **(30%)**
  - Use subjective human evaluation of the generated images in order to evaluate the quality of generated images thoroughly. Basically, the default score is based on private leaderboard of competition. We will modify the score only if quality of generated images are much better or worse than the score. Therefore, instead of being the winner in the competition, you have to make sure the generated images have reasonable quality.
- Report. **(20%)**

The final report should contain following points:

1. Pick 5 descriptions from testing data and generate 5 images with different noise  $z$  respectively.
2. Models you tried during competition. Briefly describe the main idea of the model and the reason you chose that model.
3. List the experiment you did. For example, data augmentation, hyper-parameters tuning, architecture tuning, optimizer tuning, and so on.
4. Anything worth mentioning. For example, how to pre-train the model.

## Report and inference images

Submit the link of Google Drive containing report, model and 819 inference images. Please name the report as `DL_comp3_{Your Team name}_report.ipynb`. and code of trainable model as `DL_comp3_{Your Team name}_model.ipynb`. . Moreover, please place inference images under the folder called `inference` , compress that folder with the other two notebook, and then upload to Google Drive. The compressed file should be named as `DL_comp3_{Your Team name}.zip` .

```

└─ DL_comp3_{Your Team Name}.zip
   └─ DL_comp3_{Your Team Name}_report.ipynb
   └─ DL_comp3_{Your Team Name}_model.ipynb
   └─ inference
      └─ inference_0023.jpg
      └─ inference_0041.jpg
      └─ inference_0057.jpg
      └─ ...
      └─ ...

```

## What you can do

- Pre-train text encoder on other dataset.
- Use pre-trained text encoder, like general purpose word embedding, pre-trained RNN or other language model, such as BERT, ELMo and XLNet. But you are **not allowed** to use any text encoder pre-trained on 102 flowers dataset.
- Reuse the data and model from previous competitions and labs.
- Use any package under tensorflow. You **cannot** implement your model by `tensorlayer` API.

## What you should NOT do

- Use categorical labels from flower dataset in any part of model.
- Use official [Oxford-102 flower dataset](#) and other image dataset to train your GAN. Pre-trained GAN and transfer learning are prohibited as well.
- Clone others' project or use pre-trained model from other resources (you can only use general purpose word embedding or pretrained RNN, not pretrained GAN).
- Use text encoder pre-trained on 102 flowers dataset.
- Access data or backpropagation signals from testing model in `inception_score.py` and `eval_metrics.pkl`.
- Plagiarism other teams' work.

## Hints

- You can find details about text to image in [Generative Adversarial Text to Image Synthesis](#). This competition is based on this paper.
- **Data augmentation** might improve performance a lot.
- Use more complicated **loss function** to increase training stability and efficiency.
- **Pretrained RNN** might have better hidden representation for input text. Additionally, it might accelerate the training process, and also make training more stable.
  - [Learning Deep Representations of Fine-Grained Visual Descriptions](#) model proposes a better RNN architecture and corresponding loss function for text to image task. This architecture can encode text into image-like hidden representation.
- **Different architecture of conditional GAN** might generate the image with better quality or higher resolution
  - You can try with simple GAN and DCGAN first.
  - [Generative Adversarial Text to Image Synthesis](#) proposes another RNN architecture for text to image task. The author also propose another architecture of conditional GAN to generate the images with better quality.
  - [StackGAN: Text to Photo-realistic Image Synthesis with Stacked Generative Adversarial Networks](#) proposes two-stage architecture to generate more impressive images.
  - [Improved Training of Wasserstein GANs](#) improves WGAN loss on conditional GAN can improve training stability.
  - You can find other architecture of GAN in [The GAN Zoo](#).
- Check [GAN training tips](#) to obtain some GAN training tricks, including how to generate diverse images and to prevent mode collapsing. It is worth knowing that GAN is not easy to train, and those tricks are quite helpful.

Last but not least, there are plenty of papers related to text-to-image task. [Here](#) has more information about them.

## Demo

We demonstrate the capability of our model (TA80) to generate plausible images of flowers from detailed text descriptions.

```
In [36]: def visualize(idx):
```

```

fig = plt.figure(figsize=(14, 14))

for count, i in enumerate(idx):
    loc = np.where(i==index)[0][0]
    text = ''
    for word in captions[loc]:
        if id2word_dict[word] != '<PAD>':
            text += id2word_dict[word]
            text += ' '
    print(text)

    path = './inference/TA80/inference_{:04d}.jpg'.format(i)
    fake_image = plt.imread(path)

    plt.subplot(7, 7, count+1)
    plt.imshow(fake_image)
    plt.axis('off')

```

```

In [37]: data = pd.read_pickle('./dataset/testData.pkl')
captions = data['Captions'].values
index = data['ID'].values
random_idx = [23, 216, 224, 413, 713, 859, 876, 974, 1177, 1179, 1241, 2169, 2196, 2237,
              2356, 2611, 2621, 2786, 2951, 2962, 3145, 3255, 3327, 3639, 3654, 3927, 42
              4321, 4517, 5067, 5147, 5955, 6167, 6216, 6410, 6413, 6579, 6584, 6804, 69
              7049, 7160]

visualize(random_idx)

```

flower with white long white petals and very long purple stamen  
 this medium white flower has rows of thin blue petals and thick stamen  
 this flower is white and purple in color with petals that are oval shaped  
 this flower is pink and yellow in color with petals that are oval shaped  
 the flower has a large bright orange petal with pink anther  
 the flower shown has a smooth white petal with patches of yellow as well  
 white petals that become yellow as they go to the center where there is an orange sta  
 men  
 this flower has bright red petals with green pedicel as its main features  
 this flower has the overlapping yellow petals arranged closely toward the center  
 this flower has green sepals surrounding several layers of slightly ruffled pink peta  
 ls  
 the pedicel on this flower is purple with a green sepal and rose colored petals  
 this white flower has connected circular petals with yellow stamen  
 the flower has yellow petals overlapping each other and are yellow in color  
 this flower has numerous stamen ringed by multiple layers of thin pink petals  
 the petals are broad but thin at the edges with purple tints at the edges and white i  
 n the middle  
 the yellow flower has petals that are soft smooth and arranged in two layers below th  
 e bunch of stamen  
 this flower has petals that are pink and yellow with yellow stamen  
 red stacked petals surround yellow stamen and a black pistil  
 the petals of the flower are in multiple layers and are pink in yellow in color  
 this flower has a yellow center and layers of peach colored petals with pointed tips  
 this bright pink flower has several fluttery petals and a tubular center  
 this flower is white and yellow in color with petals that are rounded at the endges  
 the flower has a several pieces of yellow colored petals that looks similar to its le  
 aves  
 this flower has several light pink petals and yellow anthers  
 this flower is yellow and white in color with petals that are star shaped near the ce  
 ner  
 lavender and white pedal and yellow small flower in the middle of the pedals  
 this flower has lavender petals with maroon stripes and brown anther filaments  
 this flower has six plain pale yellow petals that alternate with three dark yellow sp  
 eckled petals



this flower has petals that are yellow with orange lines  
 the flower has petals that are orange with yellow stamen  
 this flower has a brown center surrounded by layers of long yellow petals with rounded tips  
 this flower is lavender in color with petals that are ruffled and wavy  
 this flower is blue in color with petals that have veins  
 the petals on this flower are white with yellow stamen  
 this flower has petals that are cone shaped and dark purple  
 this flower is purple and white in color and has petals that are multi colored  
 a large group of bells that are blue on this flower  
 this flower is bright purple with many petals that are roundish with pale white outer petals  
 this flower has large yellow petals and long yellow stamen on it



Unfortunately, it is not perfect. The following figure illustrates images chosen randomly, without cherry picking.

```

In [38]: DATA_PATH = './inference/TA80/'
img_path = Path(DATA_PATH).glob('*.jpg')
img_path = [str(path.resolve()) for path in img_path]
img_path = np.asarray(img_path)

idx = np.random.randint(len(captions), size=42)
idx.sort()

random_idx = []
for each in idx:
    path = img_path[each].split("/")
  
```



```

path = path[-1].replace('inference_', '')
random_idx.append(int(path.replace('.jpg', '')))

visualize(random_idx)

```

this flower has big yellow petals and yellow stamen along with a green pedicel  
 this flower has thin pink petals as its main feature  
 this flower has petals that are pink with white edges  
 this flower is white in color with petals that are ruffled on the edges  
 small oval shaped petals arranged in a circular manner having white shades in the bot  
 tom of the petals  
 this droopy white flower has a trumpet shaped blossom surrounded by several large wid  
 e slightly pointed white petals  
 the flower has bright orange petals with a yellow inner ring  
 this flower has pointy faded pink petals that are white around their edges and have d  
 eep green veins  
 this pretty flower has yellow and orange petals with a white center  
 this flower has bright yellow petals and green pedicel and sepal  
 the petals on this flower are white with yellow stamen  
 this flower is yellow in color with petals that are bunched closely together  
 the flower shown has white petals with green lines on the inside of each petal  
 the flower is disc shaped and has soft smooth and thin petals and has stamens in the  
 centre  
 this flower has yellow petals with a tad of orange and a green pedicel  
 this flower has petals that are red with yellow stamen  
 this flower has small white and peach petals as its main feature  
 the petals of the flower are red and yellow in color and have a large green leaf  
 the flower has peach and yellow petals with yellow stamen  
 this flower has many yellow and light pink petals with dark yellow anthers  
 this flower has petals that are white with a small stigma  
 this flower is white in color with petals that are oval shaped  
 this flower has a yellow petal and green sepal and pedicel  
 this water lily has a ton of yellow stamen and white petals that stand up  
 this flower is pink and white in color with petals that are closely wrapped around th  
 e ovule  
 this flower has the overlapping yellow petals arranged closely toward the center  
 this flower has white curved petals with pink and brown spots with pink speckled stam  
 en come out of the ovule  
 this is a flower with thin wide pink petals and a white pollen tube  
 a bright white and yellow flower with a dark yellow stigma  
 this flower is white and pink in color with petals that are pink near the tips  
 this flower is white and pink in color with petals that are pink near the tips  
 this flower is pink and white in color with petals that have a pink stripe in the cen  
 ter  
 this flower has petals that are pink with a yellow center  
 this flower has a very tall pistol and pink petals that have white streaks through th  
 em  
 this flower is yellow and white in color with petals that are oval shaped  
 a flower with purple petals and dark purple pistils and anther filaments  
 flower petals are rounde in shape they are light pink in color  
 this flower is orange in color with petals that are ruffled and darker near the cente  
 r  
 petals are rounded in shape they are orange in color

