# DataLab Cup 3: Reverse Image Caption

Shan-Hung Wu & DataLab
Fall 2025

# 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

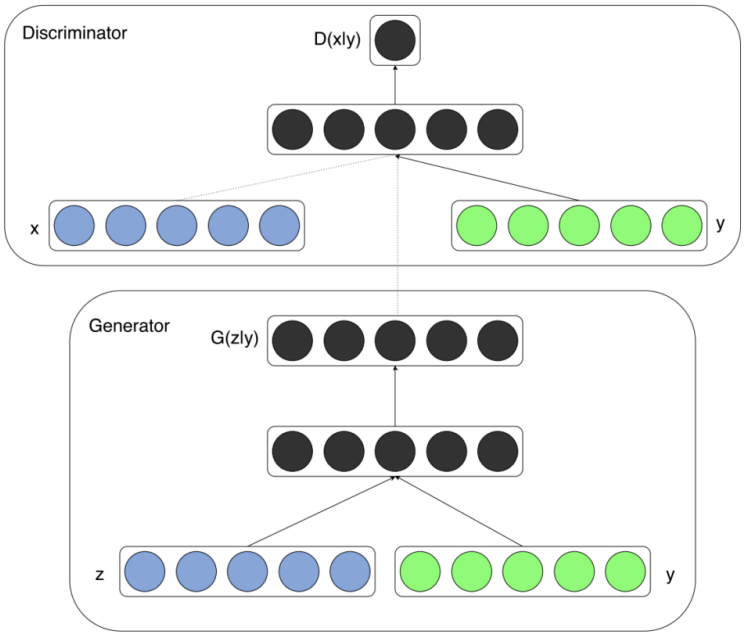
- 7370 images as training set, where each images is annotated with at most 10 texts.
- 819 texts for testing. You must generate 1 64x64x3 image for each text.

## 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.

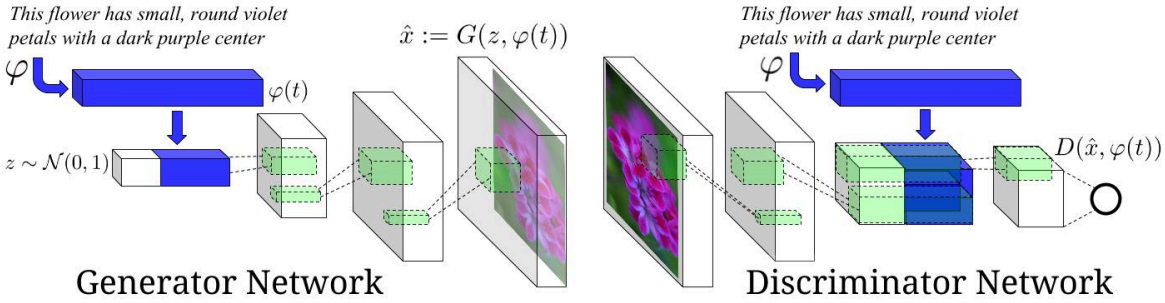
Generative adversarial nets can be extended to a conditional model if both the generator and discriminator are conditioned on some extra information $y$. We can perform the conditioning by feeding $y$ into both the discriminator and generator as additional input layer.




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 [2]:
```
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 [3]:
```
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)
```

```
1 Physical GPUs, 1 Logical GPUs
```

# 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 puntuation 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 [4]:
```
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 [5]:
```
def sent2IdList(line, MAX_SEQ_LENGTH=20):
    MAX_SEQ_LIMIT = MAX_SEQ_LENGTH
    padding = 0

    # data preprocessing, remove all puntuation 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 common word with <R
    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', '542
7', '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 a dataframe with attribute 'ID' and 'Captions', which contains testing data.

In [6]:
```
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 [7]:
```
df.head(5)
```

Out[7]:

| ID | Captions | ImagePath |
|---|---|---|
| **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 [8]:
```
# 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(filenames, batch_size, data_generator):
    # load the training data into two NumPy arrays
    df = pd.read_pickle(filenames)
```

```
    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 [9]:
```
BATCH_SIZE = 64
dataset = dataset_generator(data_path + '/text2ImgData.pkl', BATCH_SIZE, training_data_generator)
```

```
C:\Users\USER\AppData\Local\Temp\ipykernel_18420\3313815207.py:28: 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)
```

# 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 [10]:
```
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_SIZE
    """
    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_DIM'])
        # 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 [11]:
```
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 [12]:
```
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 [27]:
```
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': 10,                           # number of epoch for demo
    'N_SAMPLE': num_training_sample,         # size of training data
    'CHECKPOINTS_DIR': './checkpoints/demo', # checkpoint path
    'PRINT_FREQ': 1                          # printing frequency of loss
}
```

In [28]:
```
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 [29]:
```
# This method returns a helper function to compute cross entropy loss
cross_entropy = tf.keras.losses.BinaryCrossentropy(from_logits=True)
```

In [30]:
```
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 [31]:
```
# 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 [32]:
```
# 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 [33]:
```
@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, stddev=1.0)

    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 [34]:
```
@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 [35]:
```
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 [36]:
```
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 senteces during training, which is more convenient for us to evaluate the quality of generated image.

In [37]:
```
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'])).astype(np.float3
sample_sentence = ["the flower shown has yellow anther red pistil and bright red petals."] * int(sample_
                  ["this flower has petals that are yellow, white and purple and has dark lines"] * int(
                  ["the petals on this flower are white with a yellow center"] * int(sample_size/ni) + \
                  ["this flower has a lot of small round pink petals."] * int(sample_size/ni) + \
                  ["this flower is orange in color, and has petals that are ruffled and rounded."] * int
                  ["the flower has yellow petals and the center of it is brown."] * int(sample_size/ni) +
                  ["this flower has petals that are blue and white."] * int(sample_size/ni) +\
                  ["these white flowers have petals that start off white in color and end in a white tow

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

```
C:\Users\USER\AppData\Local\Temp\ipykernel_18420\646496249.py:3: DeprecationWarning: `np.int` is a deprec
ated alias for the builtin `int`. To silence this warning, use `int` by itself. Doing this will not modif
y 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.h
tml#deprecations
  caption = caption.astype(np.int)
```

# Training

In [38]:
```python
if not os.path.exists('samples/demo'):
    os.makedirs('samples/demo')
```

In [39]:
```python
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_per_epoch,
                                                                     d_total_loss/steps_per_epoch))
        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(epoch))
```

In [40]:
```python
train(dataset, hparas['N_EPOCH'])
```

```
Epoch 1, gen_loss: 0.4399, disc_loss: 1.1372
Time for epoch 1 is 11.8771 sec
Epoch 2, gen_loss: 0.4796, disc_loss: 1.0797
Time for epoch 2 is 9.6580 sec
Epoch 3, gen_loss: 0.6088, disc_loss: 0.8976
Time for epoch 3 is 9.8458 sec
Epoch 4, gen_loss: 1.0814, disc_loss: 0.5359
Time for epoch 4 is 9.7086 sec
Epoch 5, gen_loss: 1.8563, disc_loss: 0.2216
Time for epoch 5 is 9.5148 sec
Epoch 6, gen_loss: 1.7559, disc_loss: 0.2505
Time for epoch 6 is 9.7492 sec
Epoch 7, gen_loss: 2.3307, disc_loss: 0.1536
Time for epoch 7 is 9.5058 sec
Epoch 8, gen_loss: 2.5565, disc_loss: 0.1437
Time for epoch 8 is 9.6310 sec
Epoch 9, gen_loss: 2.6744, disc_loss: 0.1344
Time for epoch 9 is 9.7224 sec
Epoch 10, gen_loss: 3.1997, disc_loss: 0.1027
Time for epoch 10 is 9.3196 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 [41]:
```
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 [42]:
```
testing_dataset = testing_dataset_generator(hparas['BATCH_SIZE'], testing_data_generator)
```

```
C:\Users\USER\AppData\Local\Temp\ipykernel_18420\466323065.py:12: DeprecationWarning: `np.int` is a depre
cated alias for the builtin `int`. To silence this warning, use `int` by itself. Doing this will not modi
fy any behavior and is safe. When replacing `np.int`, you may wish to use e.g. `np.int64` or `np.int32` t
o specify the precision. If you wish to review your current use, check the release note link for addition
al information.
Deprecated in NumPy 1.20; for more details and guidance: https://numpy.org/devdocs/release/1.20.0-notes.h
tml#deprecations
  caption = caption.astype(np.int)
```

In [43]:
```
data = pd.read_pickle('./dataset/testData.pkl')
captions = data['Captions'].values

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

## Inferece

In [44]:
```
if not os.path.exists('./inference/demo'):
    os.makedirs('./inference/demo')
```

In [45]:
```
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'])).astype(np.fl

    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_image[i].numpy()*0.5

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

In [46]:
```
checkpoint.restore(checkpoint_dir + '/ckpt-1')
```

Out[46]:
```
<tensorflow.python.checkpoint.checkpoint.CheckpointLoadStatus at 0x281bf4f9160>
```

In [47]:
```
inference(testing_dataset)
```

```
Time for inference is 2.6524 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 exmaple, run following comment **`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

- 2025/11/13(Thur) competition announced.
- 2025/12/9(Tue) 12:00(TW) competition deadline.
- 2025/12/18(Sun) 23:59(TW) report deadline.
- 2025/12/14(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 number}_report.ipynb.` and code of trainable model as `DL_comp3_{Your Team number}_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 number}.zip` .

```
└── DL_comp3_{Your Team Number}.zip
    ├── DL_comp3_{Your Team Number}_report.ipynb
    ├── DL_comp3_{Your Team Number}_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 resouces(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](https://arxiv.org/pdf/1612.03242.pdf) 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 [50]:
```
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_iamge = plt.imread(path)

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

In [51]:
```
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, 4262,
              4321, 4517, 5067, 5147, 5955, 6167, 6216, 6410, 6413, 6579, 6584, 6804, 6988,
              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 stamen
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 petals
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 in the middle
the yellow flower has petals that are soft smooth and arranged in two layers below the 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 leaves
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 cener
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 speckled 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 pedals that are roundish whth pale white outer petals
this flower has large yellow petals and long yellow stamen on it
this flower has spiky blue petals and a spiky black stigma on it
this flower is purple and yellow in color with petals that are oval shaped
the flower has petals that are large and pink with yellow anther

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

In [53]:
```
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:
    # 提取文件名（僅最後一部分）
    filename = Path(img_path[each]).name  # 使用 pathlib 提取文件名
    # 去掉文件名中的 'inference_' 和 '.jpg'，並轉換為整數
    num = int(filename.replace('inference_', '').replace('.jpg', ''))
    random_idx.append(num)

visualize(random_idx)
```

this flower has two types of petals broad and white on the bottom and long purple and thin on top
the flower shown has purple and white petals and several white anthers
a flower with yellow petals accompanied by yellow pistils and anther filaments
the petals on this flower are yellow with white fringes
this flower has a yellow petal and green sepal and pedicel
the pedicel on this flower is purple with a green sepal and rose colored petals
this flower has petals that are red with yellow stamen
this trumpet shaped flower has several conjoined white petals with ruffled edges
this trumpet shaped flower has several conjoined white petals with ruffled edges
the flower shown has green sepal and pedicel with white petals
the flower has large red petals that are oval shaped and long
this tear drop shaped flower has one red pink petal with it s ombre pink stigma sitting at the top
the flower shown has pink and white petals with green pedicel
this flower is orange and red in color with petals that are multi colored
flowers are alternately shaped they are light pink in color
this flat flower has five light blue petals and a yellow core
this light purple flower has one continuous petal and a matching color stigma with light pink anthers
leaves are green in color petals are beige in color
the flower is pink with petals that are soft smooth and separately arranged around stamens in different l
ayers
the petals of the flower are in multiple layers and are pink in yellow in color
a light orange petaled flower set in an overapping circular pattern
this yellow flower has four tapered petals and a thin green pistil similar to the color of the stem
a flower with ragged white and purple petals and with a cluster of white stamen
the petals of this flower are pink with a short stigma
the flower has thin purple spike like petals coming from a dark round receptacle
this flower has a wide pistol with numerous yellow petals that have orange streaks down the center
this flower has thin yellow petals as its main feature
this flower has numerous yellow stamen and wide ruffled white feathers
the petals on this flower are white with yellow stamen
the small lavender flower has many small arrow shaped lavender petals
this large flower has sharp layer of yellow leaves with numerous anthers in disc in the center
this is a white flower with long oval petals
this flower is pink and white in color with petals that are pointed at the tips
the flower has petals that are white with yellow anthers
the petals of this flower are yellow with a short stigma
the flower petals are alternately arranged and are light pink in color
this flower has large white flowers and a large yellow stigma
this flower is purple and yellow in color with petals that are lighter around the edges
this flower has petals that are purple with dark dots
many clusters in a long shape with purple bell shaped flowers with white and red spots in the center
this flower is red in color with petals that are pointed at the tips
this flower is yellow and pink in color with petals that are ruffled and bunched together