CycleGAN

July 30, 2019

In [1]:
```
!pip install -q git+https://github.com/tensorflow/examples.git
```

In [2]:
```
!pip install -q tensorflow-gpu==2.0.0-beta1
import tensorflow as tf
```

In [3]:
```
from __future__ import absolute_import, division, print_function, unicode_literals

# import tensorflow_datasets as tfds
from tensorflow_examples.models.pix2pix import pix2pix

import os
import time
import matplotlib.pyplot as plt
from IPython.display import clear_output

# tfds.disable_progress_bar()
AUTOTUNE = tf.data.experimental.AUTOTUNE
```

In [4]:
```
from PIL import Image
import numpy
from scipy.misc import toimage

# LOAD TRAINING PHOTOS
folder = 'train_photos/'
file_names = [f for f in os.listdir(folder) if os.path.isfile(os.path.join(folder, f))]
print("Loading {0} photo training images...".format(len(file_names)))
train_photos = []
labels = []
for file_name in file_names:
    image = Image.open(folder + '/' + file_name)
    train_photos.append(numpy.array(image))
    labels.append(0)
print("Successfully loaded training photos!\n")
train_photos_ds = tf.data.Dataset.from_tensor_slices((train_photos, labels))

# LOAD TRAINING SKETCHES
folder = 'train_sketches/'
file_names = [f for f in os.listdir(folder) if os.path.isfile(os.path.join(folder, f))]
```

```
        print("Loading {0} sketch training images...".format(len(file_names)))
        train_sketches = []
        labels = []
        for file_name in file_names:
            image = Image.open(folder + '/' + file_name)
            train_sketches.append(numpy.array(image))
            labels.append(1)
        print("Successfully loaded training sketches!\n")
        train_sketches_ds = tf.data.Dataset.from_tensor_slices((train_sketches, labels))

        # LOAD TEST PHOTOS
        folder = 'test_photos/'
        file_names = [f for f in os.listdir(folder) if os.path.isfile(os.path.join(folder, f))]
        print("Loading {0} photo testing images...".format(len(file_names)))
        test_photos = []
        labels = []
        for file_name in file_names:
            image = Image.open(folder + '/' + file_name)
            test_photos.append(numpy.array(image))
            labels.append(0)
        print("Successfully loaded testing photos!\n")
        test_photos_ds = tf.data.Dataset.from_tensor_slices((test_photos, labels))

        # LOAD TEST SKETCHES
        folder = 'test_sketches/'
        file_names = [f for f in os.listdir(folder) if os.path.isfile(os.path.join(folder, f))]
        print("Loading {0} sketch testing images...".format(len(file_names)))
        test_sketches = []
        labels = []
        for file_name in file_names:
            image = Image.open(folder + '/' + file_name)
            test_sketches.append(numpy.array(image))
            labels.append(1)
        print("Successfully loaded testing sketches!\n")
        test_sketches_ds = tf.data.Dataset.from_tensor_slices((test_sketches, labels))

        # Show first image in the array to confirm the images were loaded.
        # Image.fromarray(train_photos[0]).show()
```

```
Loading 95 photo training images...
Successfully loaded training photos!

Loading 518 sketch training images...
Successfully loaded training sketches!

Loading 5 photo testing images...
Successfully loaded testing photos!
```

```
Loading 5 sketch testing images...
Successfully loaded testing sketches!
```

```
In [5]: BUFFER_SIZE = 1000
        BATCH_SIZE = 1
        IMG_WIDTH = 256
        IMG_HEIGHT = 256
```

```
In [6]: def random_crop(image):
            cropped_image = tf.image.random_crop(image, size=[IMG_HEIGHT, IMG_WIDTH, 3])
            return cropped_image
```

```
In [7]: # normalizing the images to [-1, 1]
        def normalize(image):
            image = tf.cast(image, tf.float32)
            image = (image / 127.5) - 1
            return image
```

```
In [8]: def random_jitter(image):
            # resizing to 286 x 286 x 3
            image = tf.image.resize(image, [286, 286],method=tf.image.ResizeMethod.NEAREST_NEI

            # randomly cropping to 256 x 256 x 3
            image = random_crop(image)

            # random mirroring
            image = tf.image.random_flip_left_right(image)

            return image
```

```
In [9]: def preprocess_image_train(image, label):
            image = random_jitter(image)
            image = normalize(image)
            return image
```

```
In [10]: def preprocess_image_test(image, label):
             image = normalize(image)
             return image
```

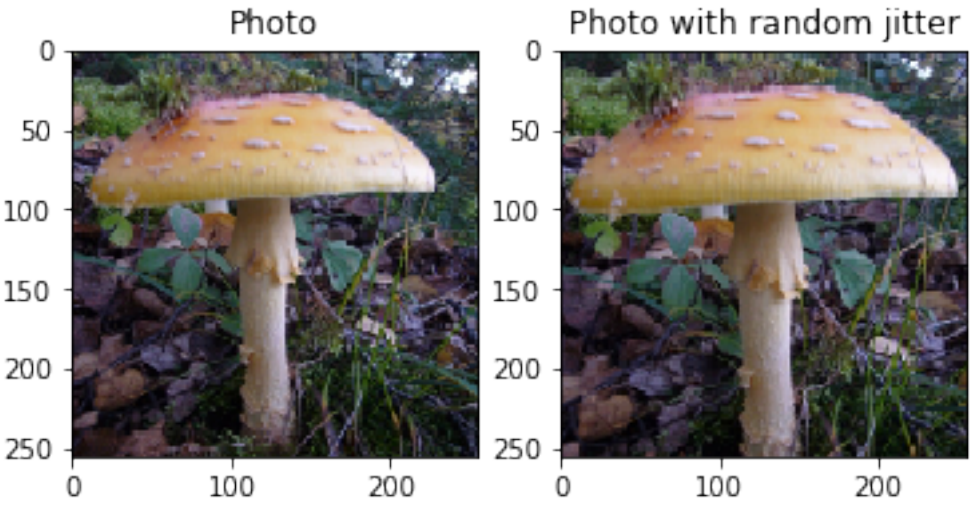
```
In [11]: train_photos_ds = train_photos_ds.map(
             preprocess_image_train,
             num_parallel_calls=AUTOTUNE).cache().shuffle(BUFFER_SIZE).batch(1)

         train_sketches_ds = train_sketches_ds.map(
             preprocess_image_train,
             num_parallel_calls=AUTOTUNE).cache().shuffle(BUFFER_SIZE).batch(1)
```

```
test_photos_ds = test_photos_ds.map(
    preprocess_image_test,
    num_parallel_calls=AUTOTUNE).cache().shuffle(BUFFER_SIZE).batch(1)

test_sketches_ds = test_sketches_ds.map(
    preprocess_image_test,
    num_parallel_calls=AUTOTUNE).cache().shuffle(BUFFER_SIZE).batch(1)
```

In [12]:
```
sample_photo = next(iter(train_photos_ds))
sample_sketch = next(iter(train_sketches_ds))
```

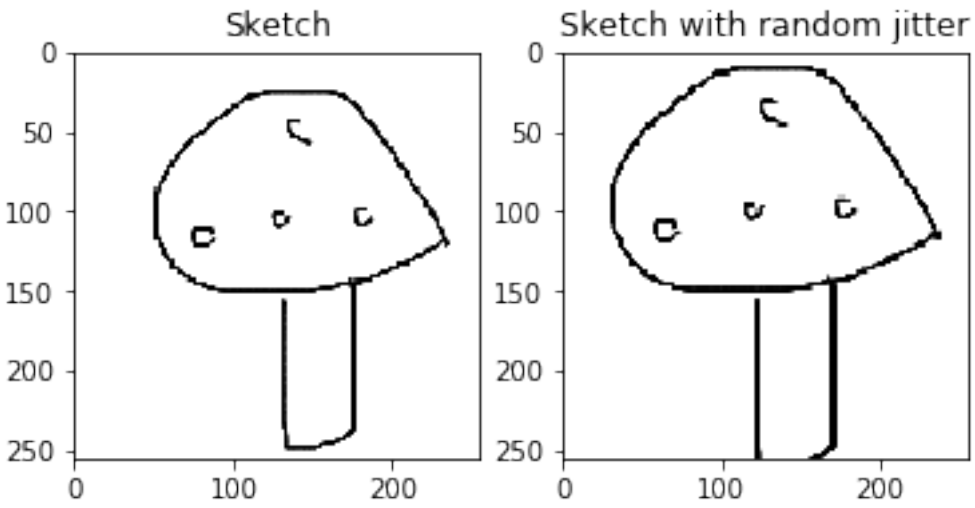
In [13]:
```
plt.subplot(121)
plt.title('Photo')
plt.imshow(sample_photo[0] * 0.5 + 0.5)

plt.subplot(122)
plt.title('Photo with random jitter')
plt.imshow(random_jitter(sample_photo[0]) * 0.5 + 0.5)
```

Out[13]: <matplotlib.image.AxesImage at 0x7fc39021cd68>



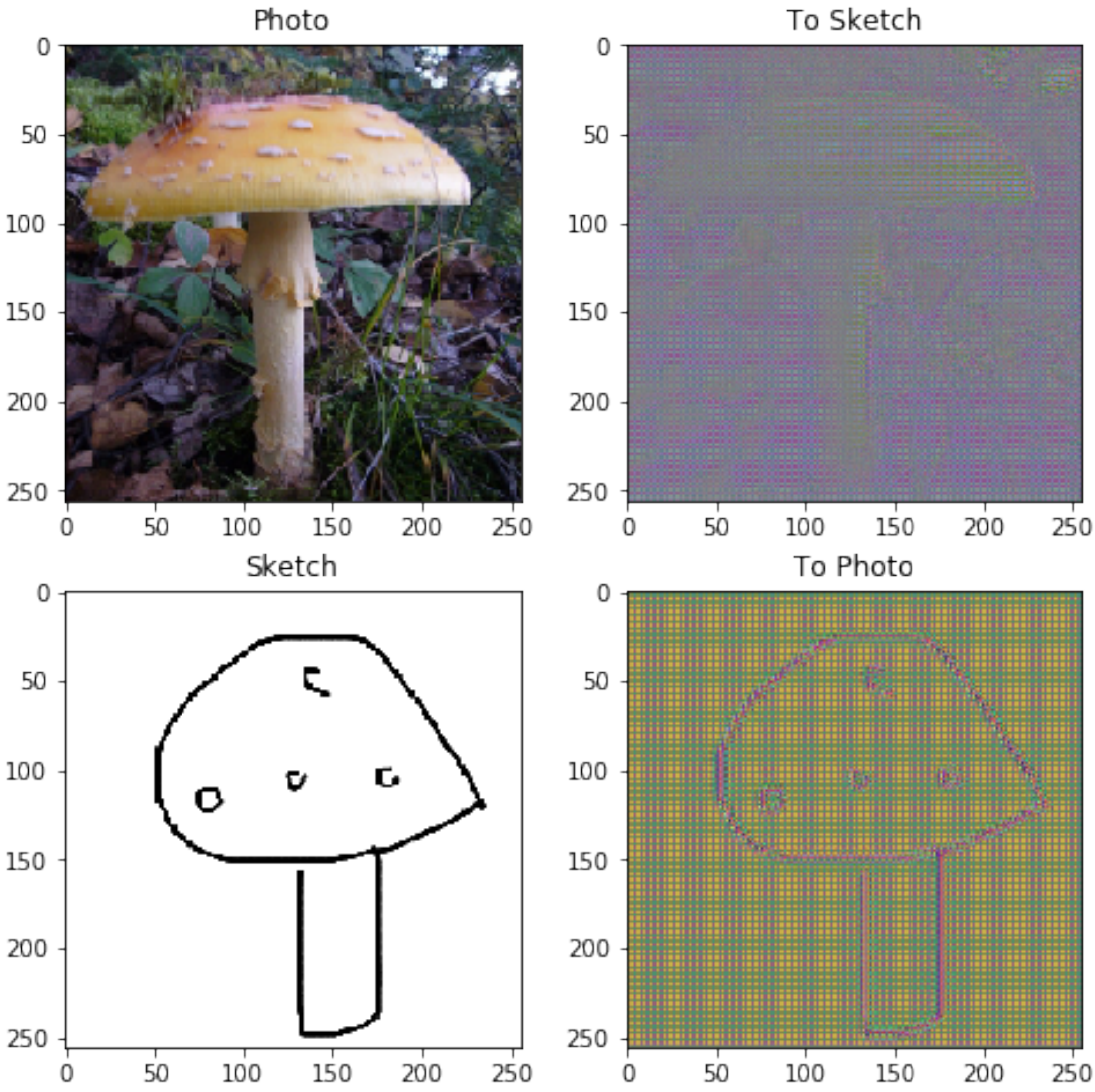
In [14]:
```
plt.subplot(121)
plt.title('Sketch')
plt.imshow(sample_sketch[0] * 0.5 + 0.5)

plt.subplot(122)
plt.title('Sketch with random jitter')
plt.imshow(random_jitter(sample_sketch[0]) * 0.5 + 0.5)
```

Out[14]: <matplotlib.image.AxesImage at 0x7fc3900ff550>

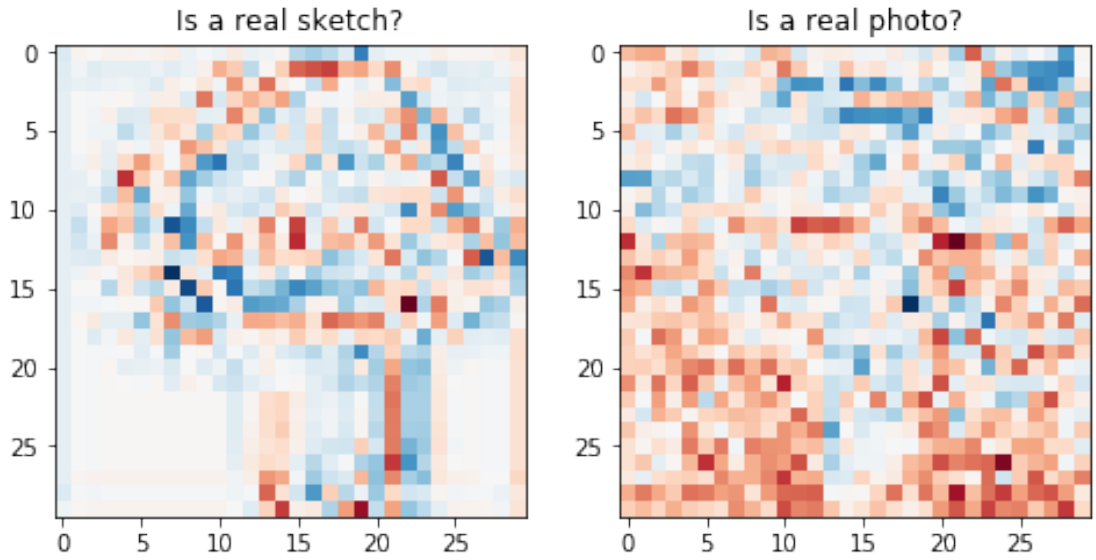


In [15]:
```
OUTPUT_CHANNELS = 3

generator_g = pix2pix.unet_generator(OUTPUT_CHANNELS, norm_type='instancenorm')
generator_f = pix2pix.unet_generator(OUTPUT_CHANNELS, norm_type='instancenorm')

discriminator_x = pix2pix.discriminator(norm_type='instancenorm', target=False)
discriminator_y = pix2pix.discriminator(norm_type='instancenorm', target=False)
```

In [16]:
```
to_sketch = generator_g(sample_photo)
to_photo = generator_f(sample_sketch)
plt.figure(figsize=(8, 8))
contrast = 8

imgs = [sample_photo, to_sketch, sample_sketch, to_photo]
title = ['Photo', 'To Sketch', 'Sketch', 'To Photo']

for i in range(len(imgs)):
    plt.subplot(2, 2, i+1)
    plt.title(title[i])
    if i % 2 == 0:
        plt.imshow(imgs[i][0] * 0.5 + 0.5)
    else:
        plt.imshow(imgs[i][0] * 0.5 * contrast + 0.5)
plt.show()
```




```
In [17]: plt.figure(figsize=(8, 8))

         plt.subplot(121)
         plt.title('Is a real sketch?')
         plt.imshow(discriminator_y(sample_sketch)[0, ..., -1], cmap='RdBu_r')

         plt.subplot(122)
         plt.title('Is a real photo?')
         plt.imshow(discriminator_x(sample_photo)[0, ..., -1], cmap='RdBu_r')

         plt.show()
```




```
In [18]: LAMBDA = 10

In [19]: loss_obj = tf.keras.losses.BinaryCrossentropy(from_logits=True)

In [20]: def discriminator_loss(real, generated):
             real_loss = loss_obj(tf.ones_like(real), real)

             generated_loss = loss_obj(tf.zeros_like(generated), generated)

             total_disc_loss = real_loss + generated_loss

             return total_disc_loss * 0.5

In [21]: def generator_loss(generated):
             return loss_obj(tf.ones_like(generated), generated)

In [22]: def calc_cycle_loss(real_image, cycled_image):
             loss1 = tf.reduce_mean(tf.abs(real_image - cycled_image))

             return LAMBDA * loss1

In [23]: def identity_loss(real_image, same_image):
             loss = tf.reduce_mean(tf.abs(real_image - same_image))
             return LAMBDA * 0.5 * loss

In [24]: generator_g_optimizer = tf.keras.optimizers.Adam(2e-4, beta_1=0.5)
         generator_f_optimizer = tf.keras.optimizers.Adam(2e-4, beta_1=0.5)

         discriminator_x_optimizer = tf.keras.optimizers.Adam(2e-4, beta_1=0.5)
         discriminator_y_optimizer = tf.keras.optimizers.Adam(2e-4, beta_1=0.5)
```

In [25]:
```
checkpoint_path = "./checkpoints/train"

ckpt = tf.train.Checkpoint(generator_g=generator_g,
                           generator_f=generator_f,
                           discriminator_x=discriminator_x,
                           discriminator_y=discriminator_y,
                           generator_g_optimizer=generator_g_optimizer,
                           generator_f_optimizer=generator_f_optimizer,
                           discriminator_x_optimizer=discriminator_x_optimizer,
                           discriminator_y_optimizer=discriminator_y_optimizer)

ckpt_manager = tf.train.CheckpointManager(ckpt, checkpoint_path, max_to_keep=5)

# if a checkpoint exists, restore the latest checkpoint.
if ckpt_manager.latest_checkpoint:
    ckpt.restore(ckpt_manager.latest_checkpoint)
    print ('Latest checkpoint restored!!')
```

In [26]:
```
EPOCHS = 200
```

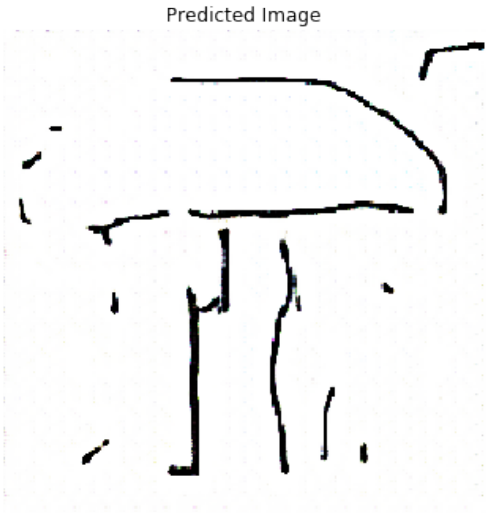
In [27]:
```
def generate_images(model, test_input):
    prediction = model(test_input)

    plt.figure(figsize=(12, 12))

    display_list = [test_input[0], prediction[0]]
    title = ['Input Image', 'Predicted Image']

    for i in range(2):
        plt.subplot(1, 2, i+1)
        plt.title(title[i])
        # getting the pixel values between [0, 1] to plot it.
        plt.imshow(display_list[i] * 0.5 + 0.5)
        plt.axis('off')
    plt.show()
```

In [28]:
```
@tf.function
def train_step(real_x, real_y):
    # persistent is set to True because the tape is used more than
    # once to calculate the gradients.
    with tf.GradientTape(persistent=True) as tape:
        # Generator G translates X -> Y
        # Generator F translates Y -> X.

        fake_y = generator_g(real_x, training=True)
        cycled_x = generator_f(fake_y, training=True)

        fake_x = generator_f(real_y, training=True)
```

```
        cycled_y = generator_g(fake_x, training=True)

        # same_x and same_y are used for identity loss.
        same_x = generator_f(real_x, training=True)
        same_y = generator_g(real_y, training=True)

        disc_real_x = discriminator_x(real_x, training=True)
        disc_real_y = discriminator_y(real_y, training=True)

        disc_fake_x = discriminator_x(fake_x, training=True)
        disc_fake_y = discriminator_y(fake_y, training=True)

        # calculate the loss
        gen_g_loss = generator_loss(disc_fake_y)
        gen_f_loss = generator_loss(disc_fake_x)

        total_cycle_loss = calc_cycle_loss(real_x, cycled_x) + calc_cycle_loss(real_y

        # Total generator loss = adversarial loss + cycle loss
        total_gen_g_loss = gen_g_loss + total_cycle_loss + identity_loss(real_y, same
        total_gen_f_loss = gen_f_loss + total_cycle_loss + identity_loss(real_x, same

        disc_x_loss = discriminator_loss(disc_real_x, disc_fake_x)
        disc_y_loss = discriminator_loss(disc_real_y, disc_fake_y)

    # Calculate the gradients for generator and discriminator
    generator_g_gradients = tape.gradient(total_gen_g_loss,
                                          generator_g.trainable_variables)
    generator_f_gradients = tape.gradient(total_gen_f_loss,
                                          generator_f.trainable_variables)

    discriminator_x_gradients = tape.gradient(disc_x_loss,
                                              discriminator_x.trainable_variables)
    discriminator_y_gradients = tape.gradient(disc_y_loss,
                                              discriminator_y.trainable_variables)

    # Apply the gradients to the optimizer
    generator_g_optimizer.apply_gradients(zip(generator_g_gradients,
                                              generator_g.trainable_variables))

    generator_f_optimizer.apply_gradients(zip(generator_f_gradients,
                                              generator_f.trainable_variables))

    discriminator_x_optimizer.apply_gradients(zip(discriminator_x_gradients,
                                                  discriminator_x.trainable_variables))

    discriminator_y_optimizer.apply_gradients(zip(discriminator_y_gradients,
                                                  discriminator_y.trainable_variables))
```

```
In [29]: for epoch in range(EPOCHS):
             start = time.time()
             print('Training epoch {} of {}.'.format(epoch + 1, EPOCHS))
             n = 0
             for image_x, image_y in tf.data.Dataset.zip((train_photos_ds, train_sketches_ds)):
                 train_step(image_x, image_y)
                 if n % 10 == 0:
                     print ('.', end='')
                 n+=1

             clear_output(wait=True)
             # Using a consistent image (sample_photo) so that the progress of the model
             # is clearly visible.
             generate_images(generator_g, sample_photo)

             if (epoch + 1) % 5 == 0:
                 ckpt_save_path = ckpt_manager.save()
                 print ('Saving checkpoint for epoch {} at {}'.format(epoch+1, ckpt_save_path))

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

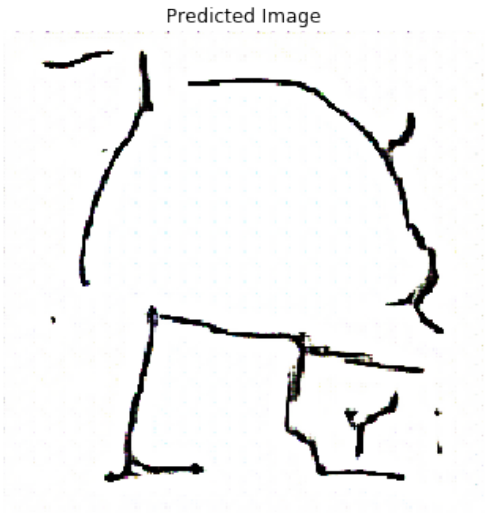
```
W0730 15:07:20.474982 140479138727744 image.py:648] Clipping input data to the valid range for
```



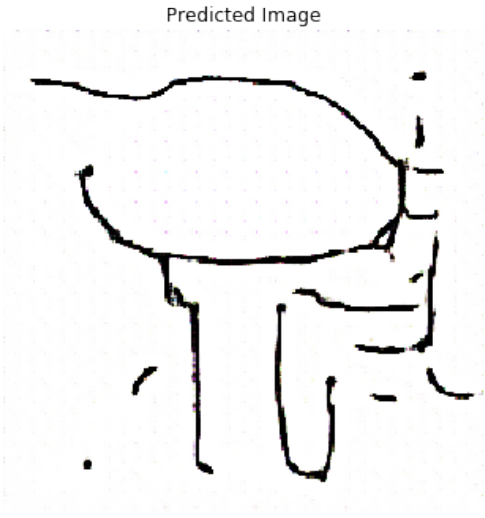

```
Saving checkpoint for epoch 200 at ./checkpoints/train/ckpt-40
Time taken for epoch 200 is 330.9848201274872 sec
```

```
In [31]: for inp in test_photos_ds.take(5):
             generate_images(generator_g, inp)
```

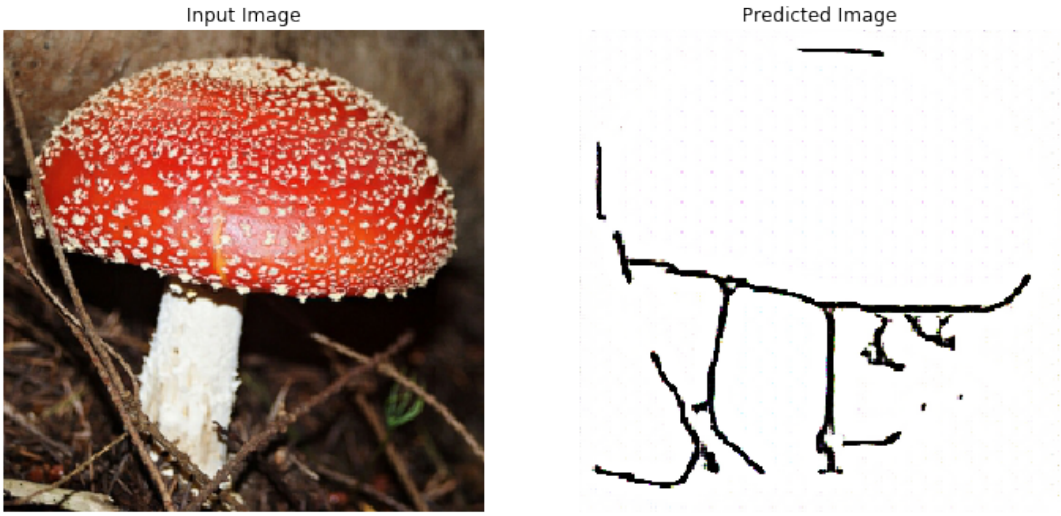
```
W0730 15:08:25.253429 140479138727744 image.py:648] Clipping input data to the valid range for
```



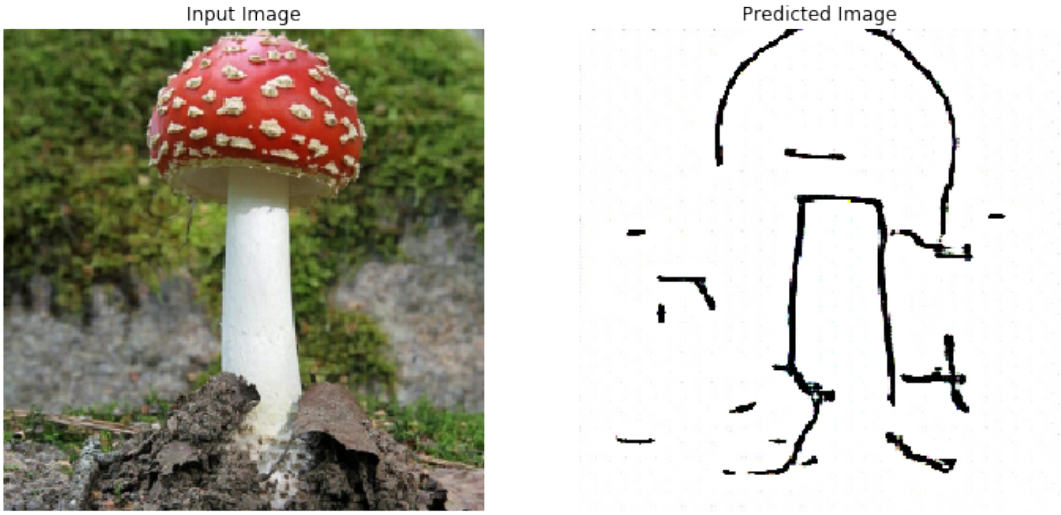

```
W0730 15:08:25.835933 140479138727744 image.py:648] Clipping input data to the valid range for
```




```
W0730 15:08:26.405554 140479138727744 image.py:648] Clipping input data to the valid range for
```



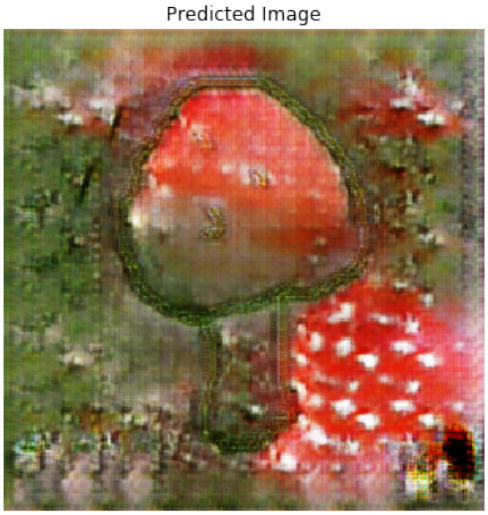

```
W0730 15:08:26.983172 140479138727744 image.py:648] Clipping input data to the valid range for
```



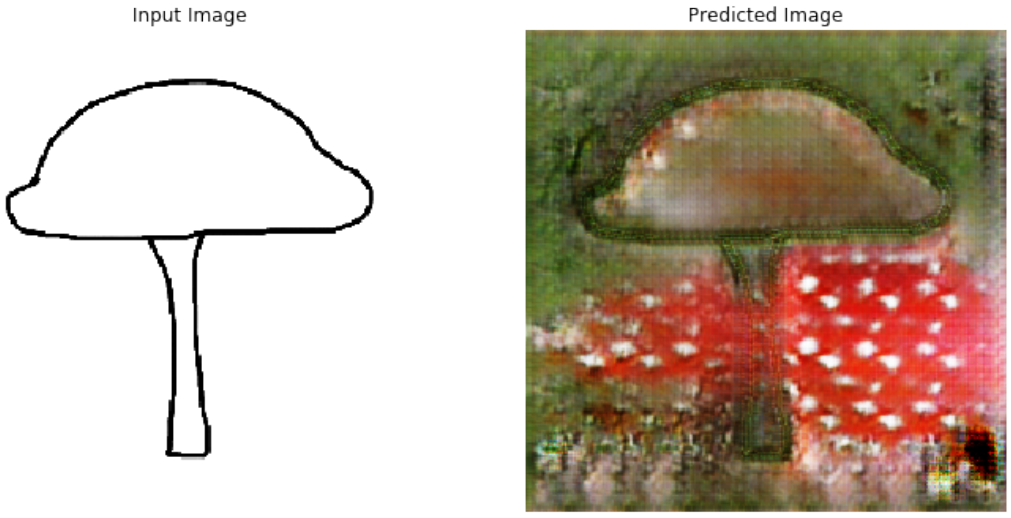

```
W0730 15:08:27.587695 140479138727744 image.py:648] Clipping input data to the valid range for
```


```
In [33]: for inp in test_sketches_ds.take(5):
             generate_images(generator_f, inp)
```

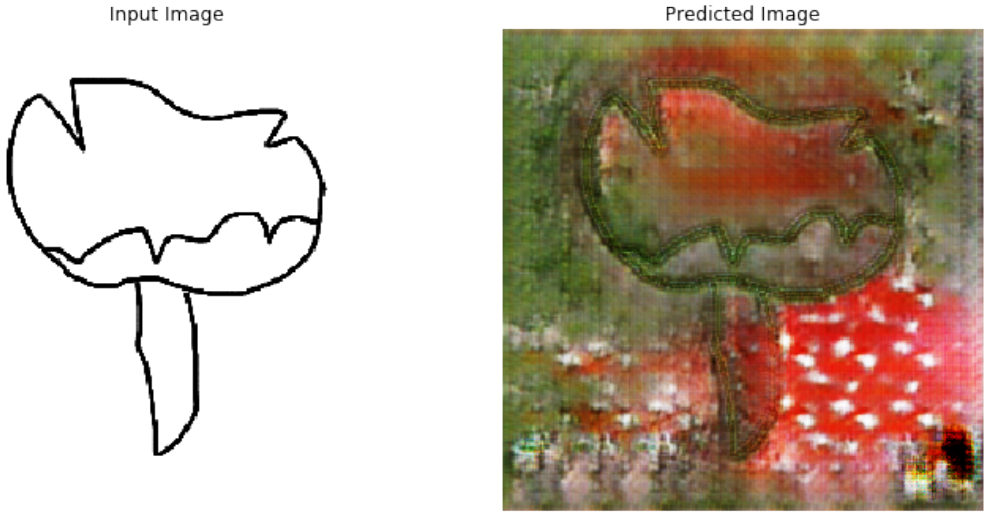
```
W0730 15:09:47.628632 140479138727744 image.py:648] Clipping input data to the valid range for
```

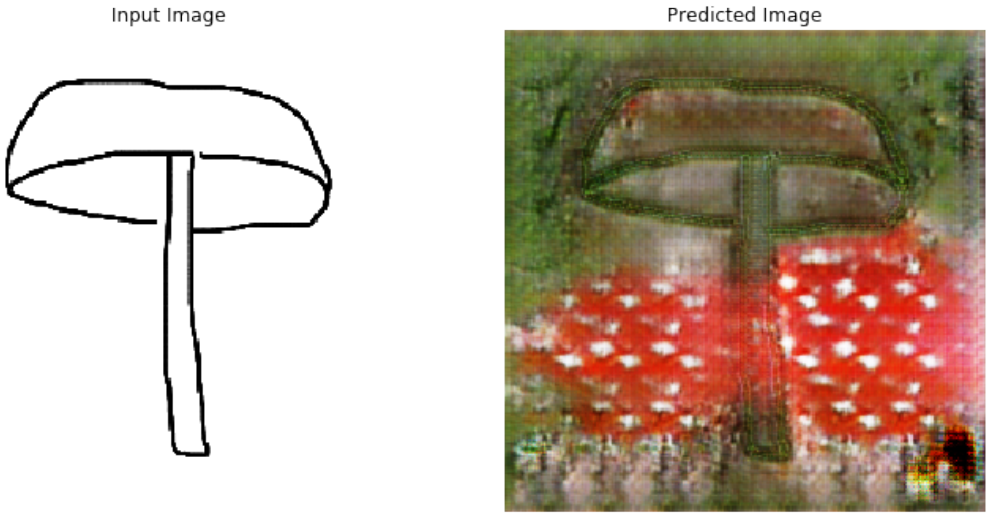



```
W0730 15:09:48.135095 140479138727744 image.py:648] Clipping input data to the valid range for
```

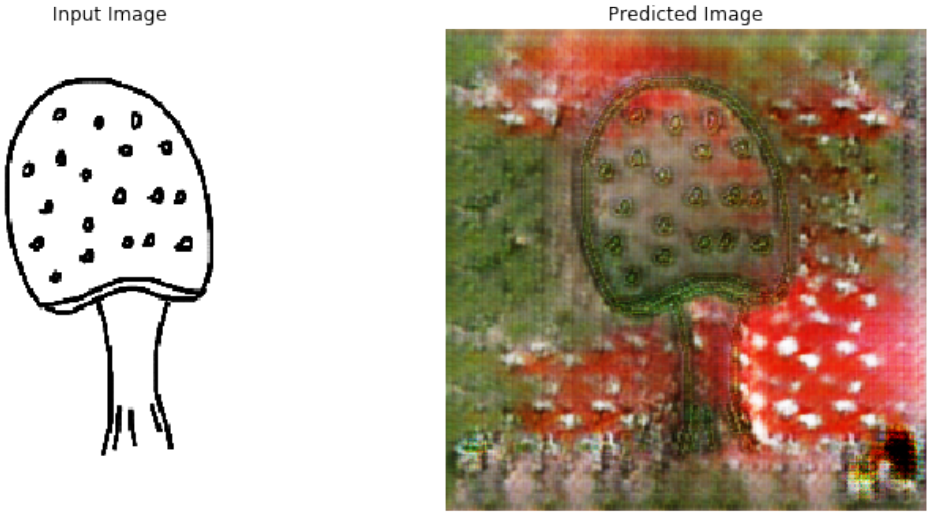



```
W0730 15:09:48.632892 140479138727744 image.py:648] Clipping input data to the valid range for
```






```
W0730 15:09:49.708443 140479138727744 image.py:648] Clipping input data to the valid range for
```



In [ ]: