Report Guide

This is a guide on how to use the report code to save your model output for the report

Running Steps

Once you are happy with how your model is performing and believe it is ready for reporting, complete the following steps:

1. Update your code base to get the latest repo changes. If on a branch, run another `git clone` on the repository in another folder to get the main branch changes without harming your branch changes. **NOTE: If you have been editing the `main` model code, be wary of pulling!**
2. Make any changes to your model as described in the **Code Changes** section of this document. This new code can be found in the `main` model in the main branch of the codebase from Step 1. Make sure to change the `YOURNAMEHERE` section of the csv name in the second last cell!
3. Run your code from start to finish, with **25 epochs**.
4. When complete, send a zip folder with the following files/folders: `dataoutput, testoutput, model_YOURNAMEHERE.csv`

Code Changes

This section will go through step by step on each changed code block. Please add any new cells and replace any edited cells in your model with this code.

1. Edited Imports

```
 import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
import tensorflow_addons as tfa
from kaggle.api.kaggle_api_extended import KaggleApi
import kaggle
import zipfile
import os
import time
from PIL import Image


import matplotlib.pyplot as plt
import numpy as np

try:
    tpu = tf.distribute.cluster_resolver.TPUClusterResolver()
    print('Device:', tpu.master())
    tf.config.experimental_connect_to_cluster(tpu)
    tf.tpu.experimental.initialize_tpu_system(tpu)
    strategy = tf.distribute.experimental.TPUStrategy(tpu)
except:
    strategy = tf.distribute.get_strategy()
print('Number of replicas:', strategy.num_replicas_in_sync)

AUTOTUNE = tf.data.experimental.AUTOTUNE

print(tf.__version__)
```

   The main imports here are `time` and `PIL`, which will be used to track training time and save images from the model, respectively.

2. Edited Training

```
 # Get time to export
class TimeHistory(keras.callbacks.Callback):
    def on_train_begin(self, logs={}):
        self.times = []

    def on_epoch_begin(self, epoch, logs={}):
        self.epoch_time_start = time.time()

    def on_epoch_end(self, epoch, logs={}):
        self.times.append(time.time() - self.epoch_time_start)

time_callback = TimeHistory()

# Get model weights to save
checkpoint_path = "training_1/cp.ckpt"
checkpoint_dir = os.path.dirname(checkpoint_path)

# Create a callback that saves the model's weights
cp_callback = tf.keras.callbacks.ModelCheckpoint(filepath=checkpoint_path,
                                                 save_weights_only=True,
                                                 verbose=1)

# Checkpoints can be reloaded and used in the model as explained here:
# https://www.tensorflow.org/tutorials/keras/save_and_load


history = cycle_gan_model.fit(
    tf.data.Dataset.zip((monet_ds, photo_ds)),
    epochs=25,
    callbacks=[time_callback, cp_callback]
)
```

There are three main changes here to look at. First, the time history class, which is used to save the time of each epoch so we can compare training times (NOTE: Since each system is different and Colab is not consistent in its training times, we're going to compare trends of training times, not direct training times). Second, there is the checkpoint saving, which will save checkpoints of your model and it's weights. If your training stops unexpectedly or it does not finish in time, you can reload the latest checkpoint and finish the training from there. Finally, we are now setting the output of the `fit` function to a variable called `history`, which will store all of the loss values which will then be exported

3. Edited Visualization

```
 # Make a directory to save the images to
if not os.path.exists('dataoutput'):
        os.makedirs('dataoutput')

_, ax = plt.subplots(10, 2, figsize=(30, 30))

# Make sure to get the same images every time
d = 'imgdata/gan-getting-started/photo_jpg'
for i, img_path in enumerate(sorted(os.listdir(d))[0:10]):
    # Get the full path and load the image
    img_path  = full_path = os.path.join(d, img_path)
    img = tf.io.read_file(img_path)
    img = decode_image(img)
    img = tf.expand_dims(
        img, axis=0
    )

    prediction = monet_generator(img, training=False)[0].numpy()
    prediction = (prediction * 127.5 + 127.5).astype(np.uint8)
    img = (img[0] * 127.5 + 127.5).numpy().astype(np.uint8)

    ax[i, 0].imshow(img)
    ax[i, 1].imshow(prediction)
    ax[i, 0].set_title("Input Photo")
    ax[i, 1].set_title("Monet-esque")
    ax[i, 0].axis("off")
    ax[i, 1].axis("off")

    # Save the image
    im = Image.fromarray(img)
    im.save("dataoutput/img{}.jpeg".format(i))
    pr = Image.fromarray(prediction)
    pr.save("dataoutput/pr{}.jpeg".format(i))

plt.show()
```

The visualization code has been edited to use the same 10 images each time for consistency, and save those images to a folder called `dataoutput`.

4. New Cells
NOTE: All of the following cells should be put **below** the currently existing cells

    - Exporting epoch times and loss function values to a csv (please change the name value in `YOURNAMEHERE` to your name, first or last or both, whatever you want)

```
 # Export training data to CSV
import csv
w = csv.writer(open("model_YOURNAMEHERE.csv", "w"))
w.writerow(['Label', 'Data'])
# Write the time as well
w.writerow(['time', time_callback.times])
for key, val in history.history.items():
    w.writerow([key, val])
```

    - Running 10 external images through the model and outputting them to a folder called `testoutput`. These images are also in the repository in `imgtest`, and the `imgtest` folder should be in the same folder as the model you're testing.

```
 # Try with a set of base images
# Make a directory to save the images to
if not os.path.exists('testoutput'):
        os.makedirs('testoutput')

_, ax = plt.subplots(10, 2, figsize=(30, 30))
d = 'imgtest'
for i, img_path in enumerate(os.listdir(d)):
    # Get the full path and load the image
    img_path  = full_path = os.path.join(d, img_path)
    img = tf.io.read_file(img_path)
    img = decode_image(img)
    img = tf.expand_dims(
        img, axis=0
    )

    # Get the prediction and alter the images
    prediction = monet_generator(img, training=False)[0].numpy()
    prediction = (prediction * 127.5 + 127.5).astype(np.uint8)
    img = (img[0] * 127.5 + 127.5).numpy().astype(np.uint8)

    # Output into a graph
    ax[i, 0].imshow(img)
    ax[i, 1].imshow(prediction)
    ax[i, 0].set_title("Input Photo")
    ax[i, 1].set_title("Monet-esque")
    ax[i, 0].axis("off")
    ax[i, 1].axis("off")

    # Save the image
    im = Image.fromarray(img)
    im.save("testoutput/imgtest{}.jpeg".format(i))
    pr = Image.fromarray(prediction)
    pr.save("testoutput/prtest{}.jpeg".format(i))
```