Branch: master ▾ Find file Copy path

w251 / hw5 / README.md

stephmather Update README.md

050bf0b 2 minutes ago

1 contributor

Raw Blame History

205 lines (172 sloc) 18 KB

Homework 5: TF2 and TF1

Stephanie Mather

Introduction to Tensorflow v2

The idea of this homework is to serve as an introduction to TensorFlow. TensorFlow 2 has just been GA'ed - and it is based on Keras, which you encountered in Session 4 and (hope you agree) is much easier to use.

- Start the tf2 container: `docker run --privileged --rm -p 8888:8888 -d w251/keras:dev-tx2-4.3_b132`
- As we did before, find out associated token - e.g. note the container id and then issue `docker logs <container_id>` and find the token. Use it to connect to your tx2 via browser on port 8888
- Glance through the TF2 beginner lab. Download this notebook from the TF hub and upload it to your TX2 container. Run it to completion.
- What's the structure of the network that's being used for classification? How good is it? Based on what you learned in homework 4, can you beat it? Hint: use something like this if you need an inspiration.

Structure is an input layer that takes the 28x28 input matrix and passes it through a single dense layer of 128 features followed by a 20% dropout layer and finally an output layer that uses softmax to classifier the 10 digits. The accuracy of this CNN is ~ 97-98%

Original:

```
model = tf.keras.models.Sequential([
  tf.keras.layers.Flatten(input_shape=(28, 28)),
  tf.keras.layers.Dense(128, activation='relu'),
  tf.keras.layers.Dropout(0.2),
  tf.keras.layers.Dense(10, activation='softmax')
])

model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])
```

- Repeat for the TF2 Quickstart lab. Download / upload to TX2 and run to completion.
- Note: you'll have to make changes to the code and fix the OOM errors. Hint: what is your batch size? *Reduced batch size to 5 then 2 was the highest possible for the final model*
- Can you improve model accuracy? Hint: are your layers frozen? Improved:

```
model = tf.keras.models.Sequential([
    tf.keras.layers.Reshape(
        target_shape=[28, 28, 1],
        input_shape=(28, 28,)),
    tf.keras.layers.Conv2D(3, 32, padding='same', activation=tf.nn.relu),
    tf.keras.layers.MaxPooling2D((2, 2), (2, 2), padding='same'),
    tf.keras.layers.Conv2D(3, 64, padding='same', activation=tf.nn.relu),
    tf.keras.layers.Conv2D(3, 64, padding='same', activation=tf.nn.relu),
    tf.keras.layers.MaxPooling2D((2, 2), (2, 2), padding='same'),
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(100, activation=tf.nn.relu),
    tf.keras.layers.Dropout(rate=0.2),
  tf.keras.layers.Dense(10, activation='softmax')
])

model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])
model.fit(x_train, y_train, epochs=7)

model.evaluate(x_test,  y_test, verbose=2)
```

Accuracy of improved model is 98.5%. See beginner.html for output. Keeping the layers frozen produced a superior result. This is likely a result of the training set being too small to meet the accuracy of the original model.

Introduction to / comparison with Tensorflow v1

TF2 is cool. However, there's a lot of code still written in Tensorflow v1, and we think that some familiarity with it will still be useful. Please try to be patient and familiarize yourself with the code of this *beginner TF1* lab. In our opinion definitely rough around the edges, while at the same time representative of what you might encounter "in the wild".

The other concepts that we hope you will pick up are architectures for image classification as well as transfer learning. The two go hand in hand: there are many pre-trained models today for image classification which you can further tweak (using transfer learning) on your own data. In this lab, you will see one approach where all the layers of the original model remain fixed.

Note also that we are doing this homework on the TX2. It is powerful enough for real time inference - and even for incremental training. This will come in handy later in the class as we begin to integrate neural processing into the kinds of pipelines you saw in homework 3.

TensorFlow for Poets

In this section, we will generally follow the Tensorflow for Poets lab at the Google CodeLabs.

Please read this before attempting the lab:

- To start an interactive TensorFlow container, run `docker run --privileged --rm -p 6006:6006 -ti w251/tensorflow:dev-tx2-4.3_b132-tf1 bash` . Note the `--rm` : when you type `exit` , this container will be removed from your TX2.
- You obviously don't need to install TensorFlow in the container explicitly (as the lab instructions suggest) ; it's already installed for you as a result of the Dockerfile instructions.
- In the command above, 6006 is the port number that Tensorboard uses. Once you launch Tensorboard (step 4), you be able to connect to the Tensorboard instance by typing http://ipaddressofyourtx2:6006
- Remember to use python3 instead of regular python for all commands, since as we mentioned above, Nvidia no longer provides a TensorFlow distro for python2 as python2 is dead!
- Once you are inside the interactive container, proceed to clone the TF for poets repository and proceed with 3+ sections of lab. Make sure you do all of the optional sections, except the "next steps" section 9.
- The Jetson packs a punch; make sure you run training for 4000 steps
- When you want to make sure the container does *not* see the GPU, run it as `docker run --rm -p 6006:6006 -ti tensorflow bash` , with no privileged flag
- On x86 based systems, Nvidia provides a tool called "nvidia-smi" to monitor GPU utilization and performance in real time. On the Jetson, this tool is not yet

supported, unfortunately. But, the Jetpack has another tool, `/usr/bin/tegrastats`. Its output looks like this:

```
root@tegra-ubuntu:~# tegrastats
RAM 2586/7846MB (lfb 1x1MB) CPU [0%@960,0%@499,0%@499,0%@959,0%@960,0%@960]
EMC_FREQ 10%@665 GR3D_FREQ 53%@140 APE 150 MTS fg 0% bg 0% BCPU@41C MCPU@41C
GPU@39C PLL@41C Tboard@35C Tdiode@37.75C PMIC@100C thermal@40.2C VDD_IN
3177/3177 VDD_CPU 536/536 VDD_GPU 383/383 VDD_SOC 536/536 VDD_WIFI 0/0 VDD_DDR
575/575
RAM 2586/7846MB (lfb 1x1MB) CPU
[49%@806,10%@345,0%@345,46%@806,34%@959,26%@960] EMC_FREQ 10%@665 GR3D_FREQ
46%@140 APE 150 MTS fg 0% bg 0% BCPU@41C MCPU@41C GPU@39C PLL@41C Tboard@35C
Tdiode@37.75C PMIC@100C thermal@40.2C VDD_IN 3177/3177 VDD_CPU 536/536 VDD_GPU
383/383 VDD_SOC 536/536 VDD_WIFI 0/0 VDD_DDR 575/575
```

GPU utilization can be deduced from the value of the GR3D_FREQ variable: the higher the value, the higher the GPU utilization.

- If you experience OOM (out of memory) errors, they could be related to the fact that the current port of Tensorflow does not understand the fact that the GPU memory that it sees is actually the same as the system memory and could be used for buffering. Run the `flush_buffers.sh` script in this repo to help clear them out and re-run your tensorflow script.
- Another way to resolve the above is to add the following to the calling script:

```
config = tf.ConfigProto()
config.gpu_options.allow_growth = True

session = tf.Session(config=config, ...)
```

In the current version of the TF for poets lab, check out line 124 of label_image.py This is where you'd need to make the change, e.g. add something like:

```
  # W251 insert
  config = tf.ConfigProto()
  config.gpu_options.allow_growth = True

#  with tf.Session(graph=graph) as sess:
  with tf.Session(graph=graph, config=config) as sess:
```

- The lab has another bug when you try to use the Inception model, because the retrain.py, line 870, states:

```
resized_input_tensor_name = 'Mul:0'
```

whereas when MobileNet is used, line 912, we have

```
resized_input_tensor_name = 'input:0'
```

- Also observe lines 867 and 868 of retrain.py

```
input_width = 299
input_height = 299
```

Questions:

1. What is TensorFlow? Which company is the leading contributor to TensorFlow? *TensorFlow is a machine learning framework that allows users to conduct machine learning and deep learning in python as a front end API for high performance C++ execution on the backend. It was created by Google, who remain large contributors to the code base.*
2. What is TensorRT? How is it different from TensorFlow? *TensorRT is an SDK for optimizing deep learning and runtime for NVIDIA GPUs. It is built in the CUDA language so that it can parallelise computation across cores. TensorRT is deeply integrated with TensorFlow. TensorFlow remains the deep learning framework and TensorRT acts as an optimisation layer that optimises the deep learning graphs wherewver possible.*
3. What is ImageNet? How many images does it contain? How many classes? *ImageNet is an image database organized according to the WordNet hierarchy (currently only the nouns), in which each node of the hierarchy is depicted by hundreds and thousands of images. The famous ImageNet database used in the annual software contest, the ImageNet Large Scale Visual Recognition Challenge has 1000 classes. However, ImageNet contains over 20,000+ categories and growing.*
4. Please research and explain the differences between MobileNet and GoogleNet (Inception) architectures.*GoogLeNet was the first version of the 'inception' architecture. Inception was a leap forward in the architecture of deep learning when moved away from the user defining all the specific of each layer (and thus relying on multiple training runs to optimise the model) and instead performs multiple different transformations at the same input layer in parallel and lets the model choose which one is the most useful. The large amounts of comuptations this could result in is reduced by using 1x1 convultions to induce dimensionality reduction. More simply, GoogLeNet is an image classification pretrained CNN with 22 layers on either the ImageNet or Places365 data sets (Places365 classifies images into 365 different place categories, such as field, park, runway, and lobby). Xception is powering Google's MobileNet application which is a mobile platform based image classifier and is short hand for 'extreme inception'. Xception uses the concept on Inception but it then does a spacewise convultion, taking into account both the positional data of the features*

as well as the features themselves. It looks for correlations across a 2D space first, followed by looking for correlations across a 1D space. MobileNet is a particular accuracy/resource tradeoff that uses the gains found in the Xception architechure to create a light weight CNN that can run on mobile platforms.

5. In your own words, what is a bottleneck? *A bottleneck is a layer that forces the neural network to concentrate feature representation by having less neurons than both the layers above and below it. By limiting the space available to record feature information they can help the model's ability to generalise as well as reducing computation.*
6. How is a bottleneck different from the concept of layer freezing? *Bottlenecks force the network to be able to generalise to new samples for the same classification set. Freezing of layers and retraining is typically used for transfer learning. The first few layers of the pre-trained network are kept (through freezing) and the later layers are allowed to be modified by the new training set. This keeps the information that is common to all image processing (such as edges and surves) but allows new features to be discovered to match the classification problem at hand.*
7. In the TF1 lab, you trained the last layer (all the previous layers retain their already-trained state). Explain how the lab used the previous layers (where did they come from? how were they used in the process?) *The CNN was sourced from MobileNet. The previous layers were trained on the ImageNet dataset. It was first published by Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, Liang-Chieh Chen: "Inverted Residuals and Linear Bottlenecks: Mobile Networks for Classification, Detection and Segmentation", 2018. For the first, full MobileNet model, the entire previous model was used. Thiis was then used on the Grace Hopper photo. The MobileNet classified the picture as military uniform, which was one of the classifications in the ImageNet training set. Later on in lab a headless model was used. This was a MobileNet model that had the last classifgication layer removed. A classifcation layer was added and then the Flowers data set was used to train the model. All the previous layers were kept, i.e. the features outputed by the CNN weret eh same as the previous full model, except they were passed through to a different classification layer. Note: it is not recommended to remove Freezing from the majority of the MobileNet layers. The model was trained on a huge data set compared tot he flowers dataset and attempting to retrain is likely to loose more information that it gains. If additional tuning is required, unfreezing of some layers, but not all may be the solution.*
8. How does a low `--learning_rate` (step 7 of TF1) value (like 0.005) affect the precision? How much longer does training take? *The speed for 0.5 was quite slow but 0.005 was very slow. Training went from a few minutes to 20min+. Scripts used are below. Precision was slightly increased with the slower learning rate for the validation set, but not by much. Precision for the training set was higher for 0.0005 learning rate. Script:

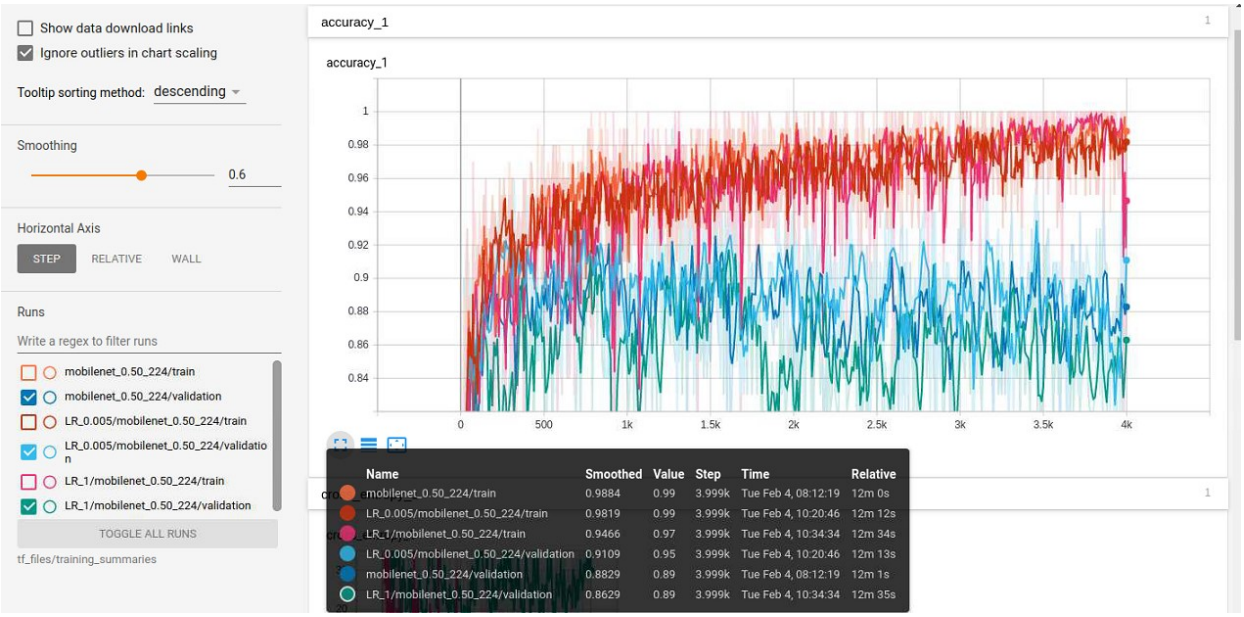
```
python3 -m scripts.retrain \
  --bottleneck_dir=tf_files/bottlenecks \
  --how_many_training_steps=4000 \
  --model_dir=tf_files/models/ \
  --summaries_dir=tf_files/training_summaries/LR_0.5/"${ARCHITECTURE}" \
  --output_graph=tf_files/retrained_graph.pb \
  --output_labels=tf_files/retrained_labels.txt \
  --architecture="${ARCHITECTURE}" \
  --image_dir=tf_files/flower_photos \
  --learning_rate=0.5

python3 -m scripts.retrain \
  --bottleneck_dir=tf_files/bottlenecks \
  --how_many_training_steps=4000 \
  --model_dir=tf_files/models/ \
  --summaries_dir=tf_files/training_summaries/LR_0.005/"${ARCHITECTURE}" \
  --output_graph=tf_files/retrained_graph.pb \
  --output_labels=tf_files/retrained_labels.txt \
  --architecture="${ARCHITECTURE}" \
  --image_dir=tf_files/flower_photos \
  --learning_rate=0.005
```

1. How about a `--learning_rate` (step 7 of TF1) of 1.0? Is the precision still good enough to produce a usable graph? The use of a high learning rate of 1 made the training of the model much quicker but there was a loss of accuracy. The accuracy was still usable - see results below.

```
python3 -m scripts.retrain \
  --bottleneck_dir=tf_files/bottlenecks \
  --model_dir=tf_files/models/ \
  --summaries_dir=tf_files/training_summaries/LR_1/"${ARCHITECTURE}" \
  --output_graph=tf_files/retrained_graph.pb \
  --output_labels=tf_files/retrained_labels.txt \
  --architecture="${ARCHITECTURE}" \
  --image_dir=tf_files/flower_photos \
  --learning_rate=1
```

Training results for 1, 0.5 and 0.005 learning rates

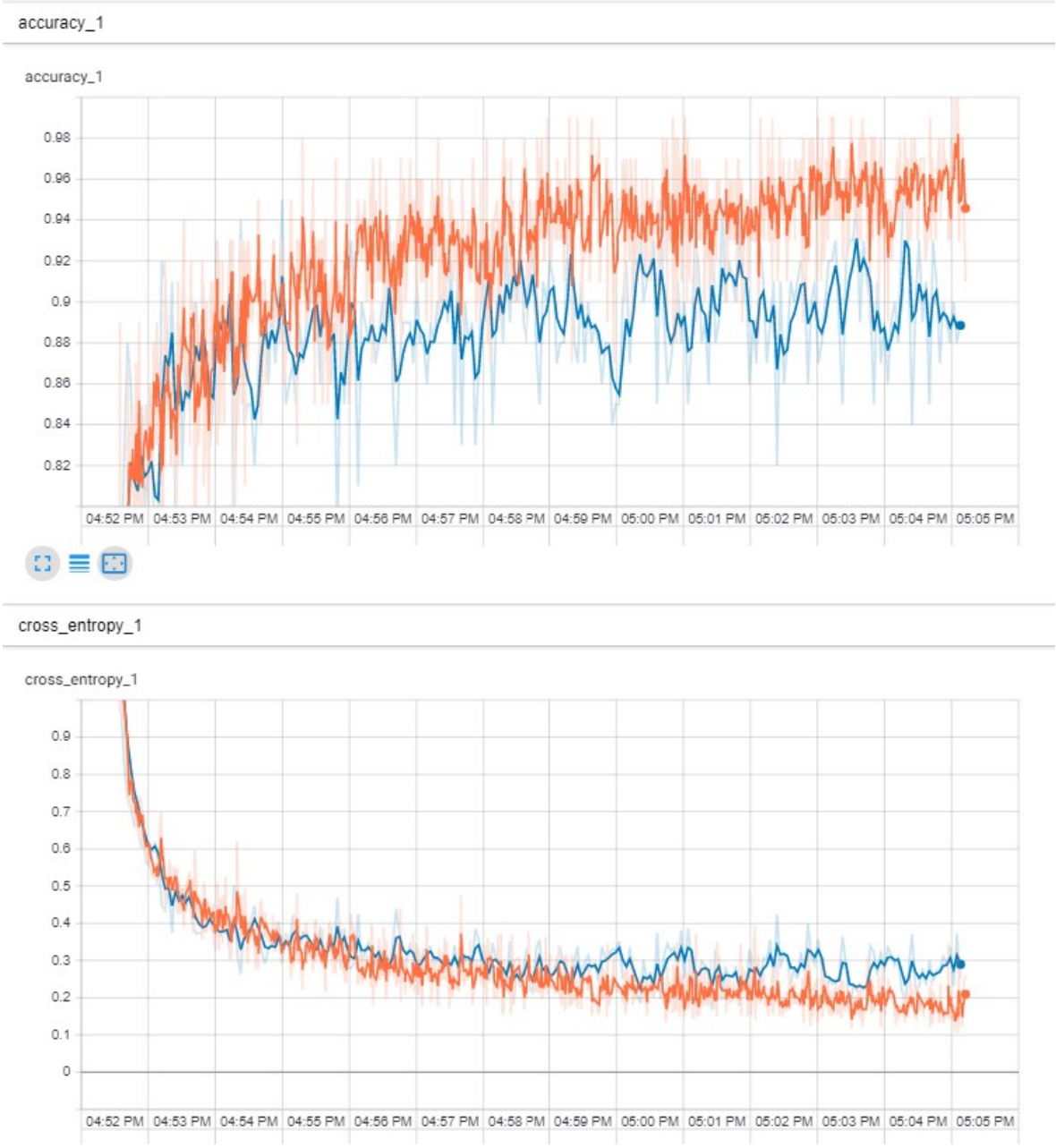


1. For step 8, you can use any images you like. Pictures of food, people, or animals work well. You can even use ImageNet images. How accurate was your model? Were you able to train it using a few images, or did you need a lot? *I trained it on a set of images I pulled from google that were either dance studios or dance stages. It did not need many images, reasonable performance was gleaned from on 150 samples in each category. The accuracy of the model was ~83% which was reasonable given the categories were hard to distinguish and some were mislabelled. For dataset urls please see dance_stage.csv and dance_studio.csv*
2. Run the TF1 script on the CPU (see instructions above) How does the training time compare to the default network training (section 4)? Why? GPU reached plateau

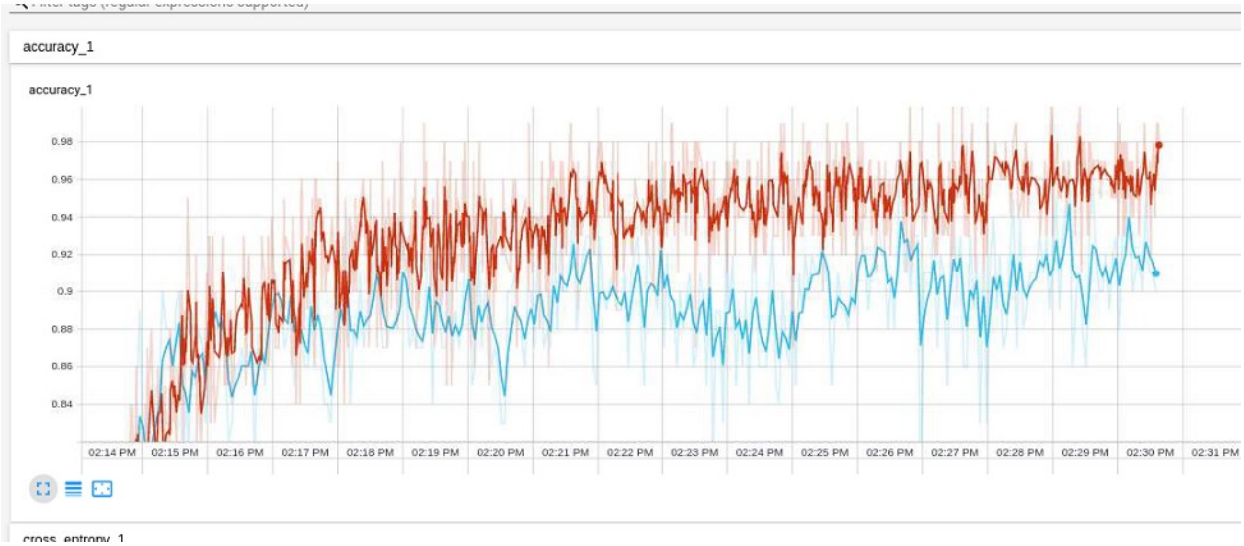
for accuracy after about 2 mins and 500 training steps

CPU reached plateau for accuracy after about 2 mins 30 sec and 700 training steps. The bottlenecks took much longer to load, increasing the overall training time.

1. Try the training again, but this time do `export ARCHITECTURE="inception_v3"` Are CPU and GPU training times different? GPU reached plateau for accuracy after about 2 mins and 1000 training steps. The bottlenecks took signifcantly longer than the mobilenet, revealling the size of the inception neural network. The load time was 14 minutes for bottlenecks on the GPU.



CPU reached plateau for accuracy after about 10 mins and 2000 training steps. The bottlenecks took much longer to load, up to 40 minutes increasing the overall training time.



1. Given the hints under the notes section, if we trained Inception_v3, what do we need to pass to replace ??? below to the label_image script? Can we also glean the answer from examining TensorBoard?

```
python -m scripts.label_image --input_layer=??? --input_height=??? --
input_width=???  --graph=tf_files/retrained_graph.pb --
image=tf_files/flower_photos/daisy/21652746_cc379e0eea_m.jpg
```

The following script addresses the bugs identified

```
python3 -m scripts.label_image --input_layer=Mul --input_height=299 --
input_width=299  --graph=tf_files/retrained_graph.pb --
image=tf_files/flower_photos/daisy/21652746_cc379e0eea_m.jpg
```

Output: Evaluation time (1-image): 7.879s

daisy (score=0.99794) sunflowers (score=0.00148) dandelion (score=0.00043) tulips (score=0.00012) roses (score=0.00003)

To turn in:

Turn in a text file or pdf with your answers to the questions above. Please note that this homework is NOT graded, credit / nocredit only.