1. Introduction

TensorFlow is a an open source library for numerical computation, specializing in machine learning applications. In this codelab, you will learn how to install and run TensorFlow on a single machine, and will train a simple classifier to classify images of flowers.

What are we going to be building?

In this lab, we will be using transfer learning, which means we are starting with a model that has been already trained on another problem. We will then be retraining it on a similar problem. Deep learning from scratch can take days, but transfer learning can be done in short order.

We are going to use the Inception v3 network. Inception v3 is a trained for the ImageNet Large Visual Recognition Challenge using the data from 2012, and it can differentiate between 1,000 different classes, like Dalmatian or dishwasher. We will use this same network, but retrain it to tell apart a small number of classes based on our own examples.

What you will learn

- How to install and run TensorFlow Docker images
- How to use Python to train an image classifier
- How to classify images with your trained classifier

What you need

- A basic understanding of Unix commands
- A fast computer running OS X or Linux
- A fair amount of time

Note: This codelab has quiet periods of downloading and training. During those times, it might be a fun idea to play with the TensorFlow Playground!

2. Setting Up

Installing TensorFlow

There are a lot of options to install TensorFlow. This codelab will cover using both Docker and not using Docker. Native installations can be slightly faster, but for this codelab, Docker is quite usable, and there is a TensorFlow image already built for you to use.

On Docker for Linux

The Docker for Linux installation instructions are here.

On Docker for OS X

For developers using OS X workstations provided by Google, the Docker runtime and container should already be available. Skip ahead to **"Running Faster."**

If you already have Docker installed, skip to "Installing/running a TensorFlow Docker Image." Otherwise, go to https://docs.docker.com/docker-for-mac/ and follow the instructions there, which should roughly be:
Download Docker for Mac.

- On the Toolbox page, find the Stable channel.
- Download Docker.dmg.
- Drag the icon into your Applications folder
- Run the Docker for Mac app.
- It will ask for root permissions to start with; allow it to do so.
- Wait until the dialog says "Docker is now up and running!".
- Run a docker command in the terminal to confirm Docker installation has worked:

```
docker run hello-world
```

Installing and Running the TensorFlow Image

On OS X, if you have not already, run the Docker for Mac app, usually placed in `/Applications/`, and which looks like this:

3. Retrieving the images

Before you start any training, you'll need a set of images to teach the network about the new classes you want to recognize. We've created an archive of creative-commons licensed flower photos to use initially.

Exit your Docker instance (ctrl-D, or type `exit`) and return to your home directory.

```
# ctrl-D if you're still in Docker and then:
cd $HOME
mkdir tf_files
cd tf_files
curl -O http://download.tensorflow.org/example_images/flower_photos.tgz
tar xzf flower_photos.tgz

# On OS X, see what's in the folder:
open flower_photos
```

After downloading 218MB, you should now have a copy of the flower photos available in your home directory.

Important: Training on this much data can take 30+ minutes on a small computer. See the next section to speed up your test.

Optional: But I'm in a hurry!

Let's reduce the number of categories we'll learn, and only tell the difference between roses and daisies for now.

You can delete them with any method; in a Unix shell, you could do this:

```
# At your normal prompt, not inside Docker
cd $HOME/tf_files/flower_photos
rm -rf dandelion sunflowers tulips

# On OS X, make sure the flowers are gone!
open .
```

Start Docker with local files available

The TensorFlow Docker image doesn't contain the flower data, so we'll make it available by linking it in virtually.

```
docker run -it -v $HOME/tf_files:/tf_files  gcr.io/tensorflow/tensorflow:latest-devel
```

Docker prompt, you can see it's linked as a toplevel directory.

```
ls /tf_files/
# Should see: flower_photos  flower_photos.tgz
```

Retrieving the training code

The Docker image you are using contains the latest GitHub TensorFlow tools, but not every last sample. You need to retrieve the full sample set this way:

```
cd /tensorflow
git pull
```

Your sample code will now be in `/tensorflow/tensorflow/examples/image_retraining/`.

4. (Re)training Inception

At this point, we have a trainer, we have data, so let's train! We will train the Inception v3 network.

As noted in the introduction, Inception is a huge image classification model with millions of parameters that can differentiate a large number of kinds of images. We're only training the final layer of that network, so training will end in a reasonable amount of time.

Start your image retraining with one big command:

```
# In Docker
python tensorflow/examples/image_retraining/retrain.py \
--bottleneck_dir=/tf_files/bottlenecks \
--how_many_training_steps 500 \
--model_dir=/tf_files/inception \
--output_graph=/tf_files/retrained_graph.pb \
--output_labels=/tf_files/retrained_labels.txt \
--image_dir /tf_files/flower_photos
```

This script loads the pre-trained Inception v3 model, removes the old final layer, and trains a new one on the flower photos you've downloaded.

ImageNet was not trained on any of these flower species, originally. However, the kinds of information that make it possible for ImageNet to differentiate among 1,000 classes are also useful for distinguishing other objects. By using this pre-trained network, we are using that information as input to the final classification layer that distinguishes our flower classes.

Important: The script can take **thirty minutes** or more to complete, depending on the speed of your machine. If you are not in a hurry, read the next section.

Optional: But NOT I'm in a hurry!

The above example iterates only 500 times. If you are training all five flower classes instead of two, you **will very likely get worse results (i.e. lower accuracy)**. To get much better results, remove the parameter `--how_many_training_steps` to use the default 4,000 iterations.

```
# In Docker
python tensorflow/examples/image_retraining/retrain.py \
--bottleneck_dir=/tf_files/bottlenecks \
--model_dir=/tf_files/inception \
--output_graph=/tf_files/retrained_graph.pb \
  --output_labels=/tf_files/retrained_labels.txt \
  --image_dir /tf_files/flower_photos
```

While You're Waiting: Bottlenecks

This section and the next are to be enjoyed while your classifier is training.

The first phase analyzes all the images on disk and calculates the bottleneck values for each of them. What's a bottleneck?

The Inception v3 model is made up of many layers stacked on top of each other (see a picture of it in this paper). These layers are pre-trained and are already very valuable at finding and summarizing information that will help classify most images. For this codelab, you are training only the last layer; the previous layers retain their already-trained state.

A 'Bottleneck,' then, is an informal term we often use for the layer just before the final output layer that actually does the classification.

Every image is reused multiple times during training. Calculating the layers behind the bottleneck for each image takes a significant amount of time. By caching the outputs of the lower layers on disk, they don't have to be repeatedly recalculated. By default, they're stored in the `/tmp/bottleneck` directory. If you rerun the script, they'll be reused, so you don't have to wait for this part again.

While You're Waiting: Training

You will see the bottlenecks train. Once they are complete, the actual training of the final layer of the network begins.

You'll see a series of step outputs, each one showing training accuracy, validation accuracy, and the cross entropy:

- The **training accuracy** shows the percentage of the images used in the current training batch that were labeled with the correct class.
- **Validation accuracy**: The validation accuracy is the precision (percentage of correctly-labelled images) on a randomly-selected group of images from a different set.
- **Cross entropy** is a loss function that gives a glimpse into how well the learning process is progressing. (Lower numbers are better here.)

A true measure of the performance of the network is to measure its performance on a data set that is not in the training data. This performance is measured using the validation accuracy. If the training accuracy is high but the validation accuracy remains low, that means the network is overfitting, and the network is memorizing particular features in the training images that don't help it classify images more generally.

The training's objective is to make the cross entropy as small as possible, so you can tell if the learning is working by keeping an eye on whether the loss keeps trending downwards, ignoring the short-term noise.

By default, this script runs 4,000 training steps. Each step chooses 10 images at random from the training set, finds their bottlenecks from the cache, and feeds them into the final layer to get predictions. Those predictions are then compared against the actual labels to update the final layer's weights through a back-propagation process.

As the process continues, you should see the reported accuracy improve. After all the training steps are complete, the script runs a final test accuracy evaluation on a set of images that are kept separate from the training and validation pictures. This test evaluation provides the best estimate of how the trained model will perform on the classification task.

You should see an accuracy value of between 85% and 99%, though the exact value will vary from run to run since there's randomness in the training process. (If you are only training on two classes, you should expect higher accuracy.) This

number value indicates the percentage of the images in the test set that are given the correct label after the model is fully trained.

5. Using the Retrained Model

The retraining script will write out a version of the Inception v3 network with a final layer retrained to your categories to `tf_files/output_graph.pb` and a text file containing the labels to `tf_files/output_labels.txt`.

These files are both in a format that the C++ and Python image classification examples can use, so you can start using your new model immediately.

Classifying an image

Here is Python that loads your new graph file and predicts with it.

label_image.py

```
import tensorflow as tf, sys

# change this as you see fit
image_path = sys.argv[1]

# Read in the image_data
image_data = tf.gfile.FastGFile(image_path, 'rb').read()

# Loads label file, strips off carriage return
label_lines = [line.rstrip() for line
                   in tf.gfile.GFile("tf_files/retrained_labels.txt")]

# Unpersists graph from file
with tf.gfile.FastGFile("tf_files/retrained_graph.pb", 'rb') as f:
    graph_def = tf.GraphDef()
    graph_def.ParseFromString(f.read())
    _ = tf.import_graph_def(graph_def, name='')

with tf.Session() as sess:
    # Feed the image_data as input to the graph and get first prediction
    softmax_tensor = sess.graph.get_tensor_by_name('final_result:0')

    predictions = sess.run(softmax_tensor, \
             {'DecodeJpeg/contents:0': image_data})

    # Sort to show labels of first prediction in order of confidence
    top_k = predictions[0].argsort()[-len(predictions[0]):][::-1]
    for node_id in top_k:
        human_string = label_lines[node_id]
        score = predictions[0][node_id]
        print('%s (score = %.5f)' % (human_string, score))
```

This is a little clumsy to cut-and-paste, so we've made a gist for you.

Exit your Docker image, and go to `$HOME/tf_files`. Create a file called **`label_image.py`** and put the above code into it. You can use `curl` to do this for you.

```
# ctrl-D to exit Docker and then:
curl -L https://goo.gl/tx3dqg > $HOME/tf_files/label_image.py
```

Restart your Docker image:

```
docker run -it -v $HOME/tf_files:/tf_files  gcr.io/tensorflow/tensorflow:latest-devel
```

Now, run the Python file you created, first on a daisy:

```
# In Docker
python /tf_files/label_image.py /tf_files/flower_photos/daisy/21652746_cc379e0eea_m.jpg
```

And then on a rose:

```
# In Docker
python /tf_files/label_image.py /tf_files/flower_photos/roses/2414954629_3708a1a04d.jpg
```

You might warnings; they not harmful. You will then see a list of flower labels, in most cases with the right flower on top (though each retrained model may be slightly different).

You might get results like this for a daisy photo:

```
daisy (score = 0.99071)
sunflowers (score = 0.00595)
dandelion (score = 0.00252)
roses (score = 0.00049)
tulips (score = 0.00032)
```

This indicates a high confidence it is a daisy, and low confidence for any other label.

You can use `label_image.py` to choose any image file to classify, either from your downloaded collection, or new ones.

6. Optional Step: Trying Other Hyperparameters

There are several other parameters you can try adjusting to see if they help your results. The `--learning_rate` controls the magnitude of the updates to the final layer during training. If this rate is smaller, the learning will take longer, but it can help the overall precision. That's not always the case, though, so you need to experiment carefully to see what works for your case.

The `--train_batch_size` parameter controls the number of images that the script examines during one training step. Because the learning rate is applied per batch, you'll need to reduce this value if you have larger batches to get the same overall effect.

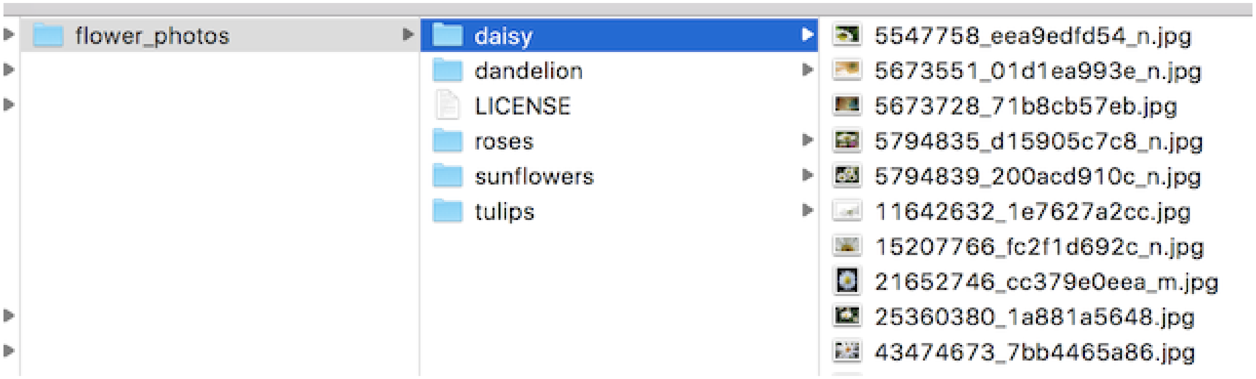
7. Optional Step: Training on Your Own Categories

After you see the script working on the flower example images, you can start looking at teaching the network to recognize categories you care about instead.

In theory, all you need to do is run the tool, specifying a particular set of sub-folders. Each sub-folder is named after one of your categories and contains only images from that category.

If you complete this step and pass the root folder of the subdirectories as the argument for the `--image_dir` parameter, the script should train the images that you've provided, just like it did for the flowers.

The classification script uses the folder names as label names, and the images inside each folder should be pictures that correspond to that label, as you can see in the flower archive:



```
flower_photos    daisy         5547758_eea9edfd54_n.jpg
                 dandelion     5673551_01d1ea993e_n.jpg
                 LICENSE       5673728_71b8cb57eb.jpg
                 roses         5794835_d15905c7c8_n.jpg
                 sunflowers    5794839_200acd910c_n.jpg
                 tulips        11642632_1e7627a2cc.jpg
                               15207766_fc2f1d692c_n.jpg
                               21652746_cc379e0eea_m.jpg
                               25360380_1a881a5648.jpg
                               43474673_7bb4465a86.jpg
```

Collect as many pictures of each label as you can and try it out!

8. Next Steps

Congratulations, you've taken your first steps into a larger world of deep learning!

You can see more about using TensorFlow at the TensorFlow website or the TensorFlow Github project. There's a list of other TensorFlow resources on the TensorFlow site, including a discussion group and whitepaper.

If you make a trained model that you want to run in production, you should also check out TensorFlow Serving, an open source project that makes it easier to manage TensorFlow projects.

This codelab is based on Pete Warden's TensorFlow for Poets blog post and this retraining tutorial.