TensorFlow for Mobile Poets

Pete Warden - pete@petewarden.com
What is this talk?
What do you need?

- OS X machine.
- Docker.
- Xcode.
- Ability to run commands in the Terminal app.
- No coding required!
Training Your Model


Deploying your model is covered in the second tutorial, at:

Picking Your Images

Just use the flowers dataset to get started.

Once you’ve had success with that, try your own categories.

- More photos are better, but a hundred each will do for a start!
- Make sure they’re similar to what you use in your app (e.g. taken on a mobile phone, shaky, badly framed, etc).
- As in life, diversity will improve results!
- Bad results? Add examples of the failures to your data set.
- Turn on random cropping, scaling, and flipping.
- Loads of GPUs? Look at full-network finetuning: https://github.com/tensorflow/models/tree/master/inception#how-to-fine-tune-a-pre-trained-model-on-a-new-task
Why Docker?

*%^%###@@&@*!*@!! Python on OS X.

Make sure you boost your VM!
Running the Training

```python
# python tensorflow/examples/image_retraining/retrain.py \
--bottleneck_dir=tf_files/bottlenecks \
--how_many_training_steps=4000 \
--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
```

Vary `how_many_training_steps` to trade off speed for accuracy.

Will take about 30 minutes.
Testing the Model

bazel build tensorflow/examples/label_image:label_image
bazel-bin/tensorflow/examples/label_image/label_image \
--graph=/tf_files/retrained_graph.pb \
--labels=/tf_files/retrained_labels.txt --output_layer=final_result \
--image=/tf_files/flower_photos/daisy/21652746_cc379e0eea_m.jpg
Now What?
Processing the Model

A great thing about deep learning is that it’s easy to transform models in a way that’s much harder for traditional programs.

We’re going to take your retrained model, and modify it to work better on mobile.
bazel build tensorflow/python/tools:optimize_for_inference
bazel-bin/tensorflow/python/tools/optimize_for_inference \
--input=/tf_files/retrained_graph.pb \
--output=/tf_files/optimized_graph.pb \
--input_names=Mul \
--output_names=final_result
Why aren’t all ops supported?

- All CPU ops can be supported on mobile.
- Some bring in unwieldy dependencies (e.g. libJPEG or file system access).
- Current mobile focuses on inference, so training-only ops don’t make sense.
- Binary footprint really matters on mobile, so skipping ops helps a lot.
- Edit tensorflow/contrib/makefile/tf_op_files.txt to change included ops.
What is BatchNorm Folding?

- During training, a varying per-channel scale is applied after each convolution.
- Once training’s done, scale doesn’t change.
- Still implemented as a separate Mul op after the Conv2D though.
- Since it’s per-channel, can be baked into the weights to reduce latency.
- Only old-style single op BatchNorm supported in this script, used in InceptionV3.
- Hope to have more advanced constant folding in the future (happens at runtime).
:quantize_graph

bazel build tensorflow/contrib/quantization/tools:quantize_graph
bazel-bin/tensorflow/contrib/quantization/tools/quantize_graph \
--input=/tf_files/optimized_graph.pb \
--output=/tf_files/rounded_graph.pb \
--output_node_names=final_result \
--mode=weights_rounded

Important part is mode=weights_rounded.
Why does Quantization Work?

- Networks trained to deal with lots of input noise.
- Done right, quantization looks like uniform noise at every layer.
- At eight bits, magnitude is below the expected input noise.
- Typically small overall loss, e.g. <1% Top-1 on ImageNet.
- Can get a lot back by retraining with a quantized forward pass.
Model Size

SqueezeNet and related work shows there’s lots of room to shrink them further.

convert_graphdef_memmapped_format

bazel build tensorflow/contrib/util:convert_graphdef_memmapped_format
bazel-bin/tensorflow/contrib/util/convert_graphdef_memmapped_format
--in_graph=/tf_files/rounded_graph.pb
--out_graph=/tf_files/mmapped_graph.pb
Isn’t this just Accounting?

- Bypasses a lot of file IO, so faster loading.
- Relieves a lot of the burden to keep pages in memory.
- We know weights are special, tells the OS it can treat them that way.
Building in Xcode

```c
// If you have your own model, modify this to the file name, and make sure
// you've added the file to your app resources too.
static NSString* model_file_name = @"mmapped_graph";
static NSString* model_file_type = @"pb";
// This controls whether we'll be loading a plain GraphDef proto, or a
// file created by the convert_graphdef_memmapped_format utility that wraps a
// GraphDef and parameter file that can be mapped into memory from file to
// reduce overall memory usage.
const bool modelUsesMemoryMapping = true;
// If you have your own model, point this to the labels file.
static NSString* labels_file_name = @"retrained_labels";
static NSString* labels_file_type = @"txt";
// These dimensions need to match those the model was trained with.
const int wanted_input_width = 299;
const int wanted_input_height = 299;
const int wanted_input_channels = 3;
const float input_mean = 128.0f;
const float input_std = 128.0f;
const std::string input_layer_name = "Mul";
const std::string output_layer_name = "final_result";
```
Why All the Parameters?

- GraphDef is just a computational graph.
- Labels, input and output names, transformations aren’t stored.
- There are alternative formats that include more info, but no single standard.
- Should get better in the future.
Demo!