Converter command line examples

This page shows how to use the TensorFlow Lite Converter in the command line.

Command-line tools

There are two approaches to running the converter in the command line.

- `tflite_convert`: Starting from TensorFlow 1.9, the command-line tool `tflite_convert` is installed as part of the Python package. All of the examples below use `tflite_convert` for simplicity.
  - Example: `tflite_convert --output_file=...`
- `bazel`: In order to run the latest version of the TensorFlow Lite Converter either install the nightly build using pip or clone the TensorFlow repository and use `bazel`.
  - Example: `bazel run //tensorflow/lite/python:tflite_convert -- --output_file=...`

Converting models prior to TensorFlow 1.9

The recommended approach for using the converter prior to TensorFlow 1.9 is the Python API. If a command line tool is desired, the `toco` command line tool was available in TensorFlow 1.7. Enter `toco --help` in Terminal for additional details on the command-line flags available. There were no command line tools in TensorFlow 1.8.

Basic examples

The following section shows examples of how to convert a basic float-point model from each of the supported data formats into a TensorFlow Lite FlatBuffers.

Convert a TensorFlow GraphDef

The follow example converts a basic TensorFlow GraphDef (frozen by freeze_graph.py) into a TensorFlow Lite FlatBuffer to perform floating-point inference. Frozen graphs contain the variables stored in Checkpoint files as Const ops.

```
curl https://storage.googleapis.com/download.tensorflow.org/models/mobilenet_v1_0.50_128_frozen.t
  | tar xzv -C /tmp
tflite_convert \
  --output_file=/tmp/foo.tflite \
  --graph_def_file=/tmp/mobilenet_v1_0.50_128/frozen_graph.pb \
  --input_arrays=input \
  --output_arrays=MobilenetV1/Predictions/Reshape_1
```

The value for `input_shapes` is automatically determined whenever possible.

Convert a TensorFlow SavedModel

The follow example converts a basic TensorFlow SavedModel into a Tensorflow Lite FlatBuffer to perform floating-point inference.

```
tflite_convert \
  --output_file=/tmp/foo.tflite \
  --saved_model_dir=/tmp/saved_model
```

SavedModel has fewer required flags than frozen graphs due to access to additional data contained within the SavedModel. The values for `--input_arrays` and `--output_arrays` are an aggregated, alphabetized list of the inputs and outputs in the SignatureDefs within theMetaGraphDef specified by `--saved_model_tag_set`. As with the GraphDef, the value for `input_shapes` is automatically determined whenever possible.

There is currently no support for MetaGraphDefs without a SignatureDef or for MetaGraphDefs that use the `assets/` directory.

Convert a tf.Keras model

The following example converts a `tf.keras` model into a TensorFlow Lite Flatbuffer. The `tf.keras` file must contain both the model and the weights.

```
tflite_convert \
  --output_file=/tmp/foo.tflite \
  --keras_model_file=/tmp/keras_model.h5
```

Quantization

Convert a TensorFlow GraphDef for quantized inference

The TensorFlow Lite Converter is compatible with fixed point quantization models described here. These are float models with `FakeQuant*` ops inserted at the boundaries of fused layers to record min-max range information. This generates a quantized inference workload that reproduces the quantization behavior that was used during training.

The following command generates a quantized TensorFlow Lite FlatBuffer from a "quantized" TensorFlow GraphDef.

```
tflite_convert \
  --output_file=/tmp/foo.tflite \
  --graph_def_file=/tmp/some_quantized_graph.pb \
  --inference_type=QUANTIZED_UINT8 \
  --input_arrays=input \
  --output_arrays=MobilenetV1/Predictions/Reshape_1 \
  --mean_values=128 \
  --std_dev_values=127
```

Use "dummy-quantization" to try out quantized inference on a float graph

In order to evaluate the possible benefit of generating a quantized graph, the converter allows "dummy-quantization" on float graphs. The flags `--default_ranges_min` and `--default_ranges_max` accept

plausible values for the min-max ranges of the values in all arrays that do not have min-max information. "Dummy-quantization" will produce lower accuracy but will emulate the performance of a correctly quantized model.

The example below contains a model using Relu6 activation functions. Therefore, a reasonable guess is that most activation ranges should be contained in [0, 6].

```
curl https://storage.googleapis.com/download.tensorflow.org/models/mobilenet_v1_0.50_128_frozen.t
  | tar xzv -C /tmp
tflite_convert \
  --output_file=/tmp/foo.cc \
  --graph_def_file=/tmp/mobilenet_v1_0.50_128/frozen_graph.pb \
  --inference_type=QUANTIZED_UINT8 \
  --input_arrays=input \
  --output_arrays=MobilenetV1/Predictions/Reshape_1 \
  --default_ranges_min=0 \
  --default_ranges_max=6 \
  --mean_values=128 \
  --std_dev_values=127
```

Specifying input and output arrays

Multiple input arrays

The flag `input_arrays` takes in a comma-separated list of input arrays as seen in the example below. This is useful for models or subgraphs with multiple inputs.

```
curl https://storage.googleapis.com/download.tensorflow.org/models/inception_v1_2016_08_28_frozen
  | tar xzv -C /tmp
tflite_convert \
  --graph_def_file=/tmp/inception_v1_2016_08_28_frozen.pb \
  --output_file=/tmp/foo.tflite \
  --input_shapes=1,28,28,96:1,28,28,16:1,28,28,192:1,28,28,64 \
  --input_arrays=InceptionV1/InceptionV1/Mixed_3b/Branch_1/Conv2d_0a_1x1/Relu,InceptionV1/Incept
  --output_arrays=InceptionV1/Logits/Predictions/Reshape_1
```

Note that `input_shapes` is provided as a colon-separated list. Each input shape corresponds to the input array at the same position in the respective list.

Multiple output arrays

The flag `output_arrays` takes in a comma-separated list of output arrays as seen in the example below. This is useful for models or subgraphs with multiple outputs.

```
curl https://storage.googleapis.com/download.tensorflow.org/models/inception_v1_2016_08_28_frozen
  | tar xzv -C /tmp
tflite_convert \
  --graph_def_file=/tmp/inception_v1_2016_08_28_frozen.pb \
  --output_file=/tmp/foo.tflite \
  --input_arrays=input \
  --output_arrays=InceptionV1/InceptionV1/Mixed_3b/Branch_1/Conv2d_0a_1x1/Relu,InceptionV1/Incept
```

Specifying subgraphs

Any array in the input file can be specified as an input or output array in order to extract subgraphs out of an input graph file. The TensorFlow Lite Converter discards the parts of the graph outside of the specific subgraph. Use graph visualizations to identify the input and output arrays that make up the desired subgraph.

The follow command shows how to extract a single fused layer out of a TensorFlow GraphDef.

```
curl https://storage.googleapis.com/download.tensorflow.org/models/inception_v1_2016_08_28_frozer
  | tar xzv -C /tmp
tflite_convert \
  --graph_def_file=/tmp/inception_v1_2016_08_28_frozen.pb \
  --output_file=/tmp/foo.pb \
  --input_shapes=1,28,28,96:1,28,28,16:1,28,28,192:1,28,28,64 \
  --input_arrays=InceptionV1/InceptionV1/Mixed_3b/Branch_1/Conv2d_0a_1x1/Relu,InceptionV1/Incept
  --output_arrays=InceptionV1/InceptionV1/Mixed_3b/concat_v2
```

Note that the final representation in TensorFlow Lite FlatBuffers tends to have coarser granularity than the very fine granularity of the TensorFlow GraphDef representation. For example, while a fully-connected layer is typically represented as at least four separate ops in TensorFlow GraphDef (Reshape, MatMul, BiasAdd, Relu...), it is typically represented as a single "fused" op (FullyConnected) in the converter's optimized representation and in the final on-device representation. As the level of granularity gets coarser, some intermediate arrays (say, the array between the MatMul and the BiasAdd in the TensorFlow GraphDef) are dropped.

When specifying intermediate arrays as `--input_arrays` and `--output_arrays`, it is desirable (and often required) to specify arrays that are meant to survive in the final form of the graph, after fusing. These are typically the outputs of activation functions (since everything in each layer until the activation function tends to get fused).

Logging

Graph visualizations

The converter can export a graph to the Graphviz Dot format for easy visualization using either the `--output_format` flag or the `--dump_graphviz_dir` flag. The subsections below outline the use cases for each.

Using `--output_format=GRAPHVIZ_DOT`

The first way to get a Graphviz rendering is to pass `GRAPHVIZ_DOT` into `--output_format`. This results in a plausible visualization of the graph. This reduces the requirements that exist during conversion from a TensorFlow GraphDef to a TensorFlow Lite FlatBuffer. This may be useful if the conversion to TFLite is failing.

```
curl https://storage.googleapis.com/download.tensorflow.org/models/mobilenet_v1_0.50_128_frozen.t
  | tar xzv -C /tmp
tflite_convert \
  --graph_def_file=/tmp/mobilenet_v1_0.50_128/frozen_graph.pb \
  --output_file=/tmp/foo.dot \
  --output_format=GRAPHVIZ_DOT \
  --input_shape=1,128,128,3 \
```

```
  --input_arrays=input \
  --output_arrays=MobilenetV1/Predictions/Reshape_1
```

The resulting `.dot` file can be rendered into a PDF as follows:

```
dot -Tpdf -O /tmp/foo.dot
```

And the resulting `.dot.pdf` can be viewed in any PDF viewer, but we suggest one with a good ability to pan and zoom across a very large page. Google Chrome does well in that respect.

```
google-chrome /tmp/foo.dot.pdf
```

Example PDF files are viewable online in the next section.

Using `--dump_graphviz_dir`

The second way to get a Graphviz rendering is to pass the `--dump_graphviz_dir` flag, specifying a destination directory to dump Graphviz rendering to. Unlike the previous approach, this one retains the original output format. This provides a visualization of the actual graph resulting from a specific conversion process.

```
curl https://storage.googleapis.com/download.tensorflow.org/models/mobilenet_v1_0.50_128_frozen.t
  | tar xzv -C /tmp
tflite_convert \
  --graph_def_file=/tmp/mobilenet_v1_0.50_128/frozen_graph.pb \
  --output_file=/tmp/foo.tflite \
  --input_arrays=input \
  --output_arrays=MobilenetV1/Predictions/Reshape_1 \
  --dump_graphviz_dir=/tmp
```

This generates a few files in the destination directory. The two most important files are `toco_AT_IMPORT.dot` and `/tmp/toco_AFTER_TRANSFORMATIONS.dot`. `toco_AT_IMPORT.dot`represents the original graph containing only the transformations done at import time. This tends to be a complex visualization with limited information about each node. It is useful in situations where a conversion command fails.

`toco_AFTER_TRANSFORMATIONS.dot` represents the graph after all transformations were applied to it, just before it is exported. Typically, this is a much smaller graph with more information about each node.

As before, these can be rendered to PDFs:

```
dot -Tpdf -O /tmp/toco_*.dot
```

Sample output files can be seen here below. Note that it is the same `AveragePool` node in the top right of each image.

before

Graph "video" logging

When `--dump_graphviz_dir` is used, one may additionally pass `--dump_graphviz_video`. This causes a graph visualization to be dumped after each individual graph transformation, resulting in thousands of files. Typically, one would then bisect into these files to understand when a given change was introduced in the graph.

Legend for the graph visualizations

- Operators are red square boxes with the following hues of red:
  - Most operators are bright red.
  - Some typically heavy operators (e.g. Conv) are rendered in a darker red.
- Arrays are octagons with the following colors:
  - Constant arrays are blue.
  - Activation arrays are gray:
    - Internal (intermediate) activation arrays are light gray.
    - Those activation arrays that are designated as `--input_arrays` or `--output_arrays` are dark gray.
  - RNN state arrays are green. Because of the way that the converter represents RNN back-edges explicitly, each RNN state is represented by a pair of green arrays:
    - The activation array that is the source of the RNN back-edge (i.e. whose contents are copied into the RNN state array after having been computed) is light green.
    - The actual RNN state array is dark green. It is the destination of the RNN back-edge updating it.