Running on mobile with TensorFlow Lite

In this section, we will show you how to use TensorFlow Lite to get a smaller model and allow you take advantage of ops that have been optimized for mobile devices. TensorFlow Lite is TensorFlow's lightweight solution for mobile and embedded devices. It enables on-device machine learning inference with low latency and a small binary size. TensorFlow Lite uses many techniques for this such as quantized kernels that allow smaller and faster (fixed-point math) models.

For this section, you will need to build TensorFlow from source to get the TensorFlow Lite support for the SSD model. At this time only SSD models are supported. Models like faster_rcnn are not supported at this time. You will also need to install the bazel build tool.

To make these commands easier to run, let's set up some environment variables:

```
export CONFIG_FILE=PATH_TO_BE_CONFIGURED/pipeline.config
export CHECKPOINT_PATH=PATH_TO_BE_CONFIGURED/model.ckpt
export OUTPUT_DIR=/tmp/tflite
```

We start with a checkpoint and get a TensorFlow frozen graph with compatible ops that we can use with TensorFlow Lite. First, you'll need to install these python libraries. Then to get the frozen graph, run the export_tflite_ssd_graph.py script from the `models/research` directory with this command:

```
object_detection/export_tflite_ssd_graph.py \
--pipeline_config_path=$CONFIG_FILE \
--trained_checkpoint_prefix=$CHECKPOINT_PATH \
--output_directory=$OUTPUT_DIR \
--add_postprocessing_op=true
```

In the /tmp/tflite directory, you should now see two files: tflite_graph.pb and tflite_graph.pbtxt. Note that the add_postprocessing flag enables the model to take advantage of a custom optimized detection post-processing operation which can be thought of as a replacement for tf.image.non_max_suppression. Make sure not to confuse export_tflite_ssd_graph with export_inference_graph in the same directory. Both scripts output frozen graphs: export_tflite_ssd_graph will output the frozen graph that we can input to TensorFlow Lite directly and is the one we'll be using.

Next we'll use TensorFlow Lite to get the optimized model by using TfLite Converter, the TensorFlow Lite Optimizing Converter. This will convert the resulting frozen graph (tflite_graph.pb) to the TensorFlow Lite flatbuffer format (detect.tflite) via the following command. For a quantized model, run this from the tensorflow/ directory:

```
bazel run -c opt tensorflow/lite/python:tflite_convert -- \
--enable_v1_converter \
--graph_def_file=$OUTPUT_DIR/tflite_graph.pb \
--output_file=$OUTPUT_DIR/detect.tflite \
--input_shapes=1,300,300,3 \
--input_arrays=normalized_input_image_tensor \
--
output_arrays='TFLite_Detection_PostProcess','TFLite_Detection_PostProcess:1','TFLite_D
```

```
 \
--inference_type=QUANTIZED_UINT8 \
--mean_values=128 \
--std_dev_values=128 \
--change_concat_input_ranges=false \
--allow_custom_ops
```

This command takes the input tensor normalized_input_image_tensor after resizing each camera image frame to 300x300 pixels. The outputs of the quantized model are named 'TFLite_Detection_PostProcess', 'TFLite_Detection_PostProcess:1', 'TFLite_Detection_PostProcess:2', and 'TFLite_Detection_PostProcess:3' and represent four arrays: detection_boxes, detection_classes, detection_scores, and num_detections. The documentation for other flags used in this command is here. If things ran successfully, you should now see a third file in the /tmp/tflite directory called detect.tflite. This file contains the graph and all model parameters and can be run via the TensorFlow Lite interpreter on the Android device. For a floating point model, run this from the tensorflow/ directory:

```
bazel run -c opt tensorflow/lite/python:tflite_convert -- \
--enable_v1_converter \
--graph_def_file=$OUTPUT_DIR/tflite_graph.pb \
--output_file=$OUTPUT_DIR/detect.tflite \
--input_shapes=1,300,300,3 \
--input_arrays=normalized_input_image_tensor \
--
output_arrays='TFLite_Detection_PostProcess','TFLite_Detection_PostProcess:1','TFLite_D
  \
--inference_type=FLOAT \
--allow_custom_ops
```

Adding Metadata to the model

To make it easier to use tflite models on mobile, you will need to add metadata to your model and also pack the associated labels file to it. If you need more information, this process is also explained in the Metadata writer Object detectors documentation

Running our model on Android

To run our TensorFlow Lite model on device, we will use Android Studio to build and run the TensorFlow Lite detection example with the new model. The example is found in the TensorFlow examples repository under `/lite/examples/object_detection`. The example can be built with Android Studio, and requires the Android SDK with build tools that support API >= 21. Additional details are available on the TensorFlow Lite example page.

Next we need to point the app to our new detect.tflite file and give it the names of our new labels. Specifically, we will copy our TensorFlow Lite flatbuffer to the app assets directory with the following command:

```
mkdir $TF_EXAMPLES/lite/examples/object_detection/android/app/src/main/assets
cp /tmp/tflite/detect.tflite \
  $TF_EXAMPLES/lite/examples/object_detection/android/app/src/main/assets
```

It's important to notice that the labels file should be packed in the model (as mentioned previously)

We will now edit the gradle build file to use these assets. First, open the `build.gradle` file `$TF_EXAMPLES/lite/examples/object_detection/android/app/build.gradle`. Comment out the model download script to avoid your assets being overwritten: `// apply from:'download_model.gradle'` ```

If your model is named `detect.tflite`, the example will use it automatically as long as they've been properly copied into the base assets directory. If you need to use a custom path or filename, open up the $TF_EXAMPLES/lite/examples/object_detection/android/app/src/main/java/org/tensorflow/demo/DetectorActivity.java file in a text editor and find the definition of TF_OD_API_MODEL_FILE. Note that if your model is quantized, the flag TF_OD_API_IS_QUANTIZED is set to true, and if your model is floating point, the flag TF_OD_API_IS_QUANTIZED is set to false. This new section of DetectorActivity.java should now look as follows for a quantized model:

```
  private static final boolean TF_OD_API_IS_QUANTIZED = true;
  private static final String TF_OD_API_MODEL_FILE = "detect.tflite";
  private static final String TF_OD_API_LABELS_FILE = "labels_list.txt";
```

Once you've copied the TensorFlow Lite model and edited the gradle build script to not use the downloaded assets, you can build and deploy the app using the usual Android Studio build process.