Converting Your TensorFlow* Model

This section describes how to run Deep Learning Model Optimizer for TensorFlow* on Linux* operating system using Python* 3. Model Optimizer converts a TensorFlow model into IR xml file and weights binary file that Inference Engine is capable to consume and run. The topic illustrates the workflow with public TensorFlow models taken from TensorFlow-Slim image classification model library (https://github.com/tensorflow/models/tree/master/research/slim#tensorflow-slim-image-classification-model-library).

The workflow consists of the following steps:

1. Prepare the inference-ready TensorFlow model
2. Prepare the checkpoint file with variable values for the model
3. Freeze the graph to make a single pb-file with both model topology and weights using the freeze_graph TensorFlow tool
4. [Optional] Explore the inputs and outputs of the model with the `summarize_graph` TensorFlow tool
5. Run the Model Optimizer Python script with pb-file to generate IR xml and weights binary files.

Prepare the inference-ready TensorFlow model

This paragraph uses a pre-trained TensorFlow model for illustrative purposes. In particular, Inception V1 pre-trained model from TensorFlow-Slim image classification model library (https://github.com/tensorflow/models/tree/master/research/slim#pre-trained-models). The following steps are specific for models from this library. For your own model they are not needed. Please go ahead and skip the Slim specific section to the next section where

For more complete information about compiler optimizations, see

common instructions will be given.

Several next sections are not specific for the Model Optimizer flow. They are general steps of TensorFlow model preparation for the inference. This section is close to existing TensorFlow tutorials and frequently reference to them.

Getting a model from Slim library

Tensorflow/models is a dedicated repository for various TensorFlow models that is separate from the main TensorFlow repository. It contains Python scripts to build networks and useful high-level infrastructure to work with models easily.

Please follow this instruction (https://github.com/tensorflow/models/tree/master/research/slim#installing-the-tf-slim-image-models-library) to get the tensorflow/models repository. Basically you need to just clone https://github.com/tensorflow/models (https://github.com/tensorflow/models) to a directory of your choice referred to as `<SLIM_DIR>`.

> **NOTE:** The TensorFlow support is an experimental feature of Model Optimizer, to make all the next steps smooth, please check out specific old version of the models repository:

```
$ cd <SLIM_DIR>
$ git checkout 09f32cea15cac4358592bec866b5ca68b0b5f7c
```

Here is an explanation: as we required TensorFlow 1.2 we will use some tools for graph manipulation from TensorFlow 1.2. These tools are not fully compatible with the new models, particularly `summarize_graph` may report an error. So by doing that checkout of a specific commit, you are getting models repository closer to TensorFlow 1.2 branch, enabling all steps from the next sections of this document.

Once you check out appropriate commit of the models repository, you need to export inference graph to the pb-file. This file will contain network topology *without weights*. Following this instruction (https://github.com/tensorflow/models/tree/master/research/slim#exporting-the-inference-graph), you need to run `<SLIM_DIR>/slim/export_inference_graph.py` (note that the path to the file is valid for recommended revision checked out earlier; for newer revisions the directory structure is changed):

```
$ cd <SLIM_DIR>/slim
$ python3 export_inference_graph.py \
    --alsologtostderr \
    --model_name=inception_v1 \
```

For more complete information about compiler optimizations, see

```
5     --output_file=/tmp/inception_v1_inf_graph.pb
```

The value of `--model_name` parameter should be one of the implemented topologies in Slim.

> **NOTE:** Not all topologies from the library are supported by the current version of Model Optimizer for TensorFlow.

`--output_file` parameter value is target file name with exported model. You can chose any directory. Later we will combine this file with an appropriate checkpoint file to get a model file ready to be fed to Model Optimizer.

Together with network scripts, this section (https://github.com/tensorflow/models/tree/master/research/slim#pre-trained-models) of Slim README file contains links to the pre-trained checkpoints. A checkpoint is a file with values for all variables for the model. Such a file is produced as a result of model training.

For this guide, you should download an archive with checkpoint for Inception V1 model. The archive contains `inception_v1.ckpt` file. Unpack it to the directory of your choice. We will refer to it as `<CKPT_DIR>`.

Preparing a custom model

The section above is specific for the Slim models. In case of a custom model, you need to export TensorFlow graph from your session to a binary pb file that contains GraphDef description and to save a checkpoint file that contains all variables values of the graph. Having these two pieces, you are able to proceed to the next section. Please refer to the TensorFlow documentation to learn how to export any graph and checkpoint file from a TensorFlow session.

Make sure that the exported graph is free of any training-specific operations. For example, TensorFlow dropout shouldn't be included into the inference graph, because it is needed only at the training stage.

The inference model should have only a single input that is represented as a `placeholder` TensorFlow operation.

Freezing inference graph

Freezing the graph is an operation that combines network GraphDef pb-file exported earlier with checkpoint file to produce a single model file. Such a file has all graph variables frozen as constants together with a graph topology. To do the

For more complete information about compiler optimizations, see

freezing you have to use a TensorFlow tool freeze_graph.py. Or if your model has been already exported with all variables converted to constants, you can skip this section completely.

Following this instruction (https://github.com/tensorflow/models/tree/master/research/slim#freezing-the-exported-graph), you need first build the tool and then run it:

```
$ cd <TF_HOME>
$ bazel build tensorflow/python/tools:freeze_graph
$ bazel-bin/tensorflow/python/tools/freeze_graph \
    --input_graph=/tmp/inception_v1_inf_graph.pb \
    --input_checkpoint=<CKPT_DIR>/inception_v1.ckpt \
    --input_binary=true \
    --output_graph=/tmp/frozen_inception_v1.pb \
    --output_node_names=InceptionV1/Logits/Predictions
```

In contrast to the most of the parameters, which are easy to set, to deduce value for `--output_node_names` you need to know the name(s) of the resulting operation(s) of your model. They are names of operations in the network that produce the desired output of the model. Usually such operations don't have other consumers in the network. The names are formed during model building and then are written to the network pb-file, e.g. `/tmp/inception_v1_inf_graph.pb` in our case. In our case, it is a single output `InceptionV1/Predictions/Reshape`. In general case, you can use `summarize_graph` TensorFlow tool to extract output nodes candidates.

Using `summarize_graph` to examine a model

TensorFlow tool `summarize_graph` is described here (https://github.com/tensorflow/tensorflow/tree/master/tensorflow/tools/graph_transforms#inspecting-graphs). The key information that you can retrieve with this tool is names of input and output nodes of the model. It is used at the graph freezing step as well as at the latest stage when we will run Model Optimizer.

You need to build the tool first:

```
$ cd <TF_HOME>
$ bazel build tensorflow/tools/graph_transforms:summar
```

It is useful to extract output node name by running the tool with not yet frozen

graph to get possible values for `--output_node_names` for `freeze_graph` tool. If we run the tool in this way:

```
$  bazel-bin/tensorflow/tools/graph_transforms/summari
```

It will print a lot of output with `--output_layer=InceptionV1/Logits /Predictions/Reshape_1` in the end. It is exactly the name that we passed to `freeze_graph` as `--output_node_names` in the previous section. So you can use the same approach for your own models.

Then it is also useful to run `summarize_graph` with already frozen model, in our case it is `/tmp/frozen_inception_v1.pb`:

```
$ bazel-bin/tensorflow/tools/graph_transforms/summariz
```

Here is the tool output, which is significantly shorter:

```
Found 1 possible inputs: (name=input, type=float(1), s
No variables spotted.
Found 1 possible outputs: (name=InceptionV1/Logits/Pre
Found 6633279 (6.63M) const parameters, 0 (0) variable
Op types used: 298 Const, 231 Identity, 114 Add, 114 M
Softmax, 1 Squeeze
To use with tensorflow/tools/benchmark:benchmark_model
bazel run tensorflow/tools/benchmark:benchmark_model -
--input_layer_shape=1,224,224,3 --output_layer=Incepti
```

You need to save the following key information about the model to use when you run Model Optimizer:

- `Found 1 possible inputs: (name=input, type=float(1), shape=[1,224,224,3])`
  - `input` is a name of input node (used in `--input` Model Optimizer parameter)
  - `float` is a type of input tensor element (used in `--data_type` Model Optimizer parameter)
  - `1,224,224,3` is a shape of input (used in `--input_shape` Model Optimizer parameter)

- `Found 1 possible outputs: (name=InceptionV1/Logits /Predictions/Reshape_1, op=Reshape)`
  - `InceptionV1/Logits/Predictions/Reshape_1` is a name of the

For more complete information about compiler optimizations, see

output (used in `--output` Model Optimizer parameter)

Running Model Optimizer

Well, we are ready to make the final step of model conversion to IR.

Go to the Deep Learning Model Optimizer for TensorFlow root directory and run:

```
$ python3 modeloptimizer/scripts/model_optimizer.py \
    --input_model=/tmp/frozen_inception_v1.pb \
    --input=input \
    --output=InceptionV1/Logits/Predictions/Reshape_1
    --data_type=float \
    --input_shape 1,224,224,3 \
    --model_name InceptionV1
```

Note that `--input_model` accepts only full path to an input model file. Relative ones like `~/models/inception` would not work for you.

It is very important to set the name of the input and output layer in `--input` and `--output` argument respectively.

After successful run of `model_optimizer.py`, in the current directory you will find: - `InceptionV1.xml` - Contains IR for the model - `InceptionV1.bin` - Contains weights referenced from the IR

After you have generated IR, run it via the appropriate sample (/node/8e4ea7df-d2cb-445a-a829-c96a1b72183b). In this case, as Inception V1 is a classification model, you need to refer to instructions for building IE samples and then running the classification sample (/node/317737f7-a00c-4517-bdd0-065952b22c67).

Working with Transforms

This is the advanced feature of the Model Optimizer. Use it if you completely understand the full workflow.

Default transform sequence

The model transformation pipeline is defined by optional `--transforms` parameter of `model_optimizer.py` script. A value for `--transform` is a string with space-separated transform names and their parameters enclosed into parenthesis. When `--transform` is not passed in command line to Model

For more complete information about compiler optimizations, see

Optimizer, the default sequence is used:

- `strip_unused_nodes(type=<data_type>;shape=<input_shape>)`
- `remove_nodes(op=Identity)`
- `remove_nodes(op=CheckNumerics)`
- `fold_constants(ignore_errors=true)`
- `fold_batch_norms`
- `strip_unused_nodes(type=<data_type>; shape=<input_shape>)`
- `remove_nodes(op=Identity)`
- `remove_nodes(op=CheckNumerics)`
- `fold_constants(ignore_errors=true)`
- `fold_batch_norms`
- `strip_unused_nodes(type=<data_type>; shape=<input_shape>)`
- `remove_nodes(op=Identity)`
- `remove_nodes(op=CheckNumerics)`
- `fold_constants(ignore_errors=true)`
- `fold_batch_norms`
- `calc_shapes(input_types=<data_type>; input_shapes=<input_shape>)`
- `create_ir(model_name=<model_name>; output_dir=<output_dir>; scale=<scale>)`

The transforms from the list are applied in the specified order. `<data_type>`, `<input_shape>`, `<model_name>`, `<output_dir>` and `<scale>` are values of corresponding Model Optimizer command line parameters (i.e. `--data_type`, `--input_shape` etc.). The last transform should be always `create_ir`.

So when you ran `model_optimizer` without `--transform` in the previous section, it was equivalent to

```
$ python3 modeloptimizer/scripts/model_optimizer.py \
    --input_model=/tmp/frozen_inception_v1.pb \
    --input=input \
    --output=InceptionV1/Logits/Predictions/Reshape_1
    --transforms="\
        strip_unused_nodes(type=float;shape=1,224,224,
        remove_nodes(op=Identity) \
        remove_nodes(op=CheckNumerics) \
        fold_constants(ignore_errors=true) \
        fold_batch_norms \
        strip_unused_nodes(type=float; shape=1,224,224
```

For more complete information about compiler optimizations, see

```
        remove_nodes(op=Identity) \
        remove_nodes(op=CheckNumerics) \
        fold_constants(ignore_errors=true) \
        fold_batch_norms \
        strip_unused_nodes(type=float; shape=1,224,224
        remove_nodes(op=Identity) \
        remove_nodes(op=CheckNumerics) \
        fold_constants(ignore_errors=true) \
        fold_batch_norms  \
        calc_shapes(input_types=float; input_shapes=1,
        create_ir(model_name=InceptionV1; output_dir=.
```

It is recommended to start with this command line and modify transform list in appropriate for your model way.

There are two kind of transforms in the list: - Model Optimizer transforms - Standard TensorFlow transforms from the Graph Transform Tool

Model Optimizer transforms

calc_shapes

- It receives two parameters: `input_types` and `input_shapes`.
- It populates the shapes of all the tensors in the graph.
- If your model has shape computations that you want to fold before the model deployment, try to mix `calc_shapes` with `fold_constants` and repeat it several times together with other transforms from the default list.

create_ir

- This transform takes a learned model from TensorFlow (protobuf file) and creates an new optimized model which holds xml and bin files (also known as IR format, for more info, see IR Notation Reference (/node/7a34decc-165c-468b-b7cc-9bf6a18496b1)). It is highly recommended to optimize this protobuf before running this transform, for example with `fold_batch_norms`.
- The arguments which this transform receives are:
  - `model_name` – a required flag. The model name and the name for the IR that will be created. Do not use any symbols except literals and numbers.
  - `ouput_dir` – a required flag. The output directory, which the IR will be created to.

For more complete information about compiler optimizations, see

- `precision` – an optional flag. For now, only FP32 is supported which is the default value.
- `scale` – an optional flag. The scale factor for the input image. For now, scale is the same for all input channels. Default value is 1.
- `batch_size` – an optional flag. Default value is 1.
- `input_format` – an optional flag. For now, RGB and BGR are supported. Default value is BGR.

Graph Transform Tool transforms

All the TensorFlow graph transform tool supported transforms are integrated and supported by the Model Optimizer. They are called directly from TensorFlow.

Review the full list of available transforms here (https://github.com/tensorflow/tensorflow/tree/master/tensorflow/tools/graph_transforms#graph-transform-tool).

See Also

- Converting a Model to Intermediate Representation (IR) (/node/fb2b41c2-10de-482a-9aa9-3c6618052935)
- Configuring TensorFlow* (/node/cdb3d960-d667-4963-aa21-d779f95ce81d)
- Configuring Model Optimizer for TensorFlow* (/node/64648cc5-b78d-47b3-88de-65afa110f90f)
- Using Sample Applications (/node/8e4ea7df-d2cb-445a-a829-c96a1b72183b)
- IR Notation Reference (/node/7a34decc-165c-468b-b7cc-9bf6a18496b1)