Request for Quotations

TensorFlow to NNEF Converter

January 2018

Notice

ALL KHRONOS SPECIFICATIONS AND OTHER DOCUMENTS (TOGETHER AND SEPARATELY, "MATERIALS") ARE BEING PROVIDED "AS IS." KHRONOS MAKES NO WARRANTIES, EXPRESSED, IMPLIED, STATUTORY OR OTHERWISE WITH RESPECT TO THE MATERIALS, AND EXPRESSLY DISCLAIMS ALL IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE.

Information furnished is believed to be accurate and reliable. However, Khronos assumes no responsibility for the consequences of use of such information or for any infringement of patents or other rights of third parties that may result from its use. No license is granted by implication or otherwise under any patent or patent rights of Khronos. Specifications mentioned in this publication are subject to change without notice. This publication supersedes and replaces all information previously supplied.

Trademarks

Khronos and NNEF, and associated logos are trademarks or registered trademarks of Khronos Group Inc. All other product names, trademarks, and/or company names are used solely for identification and belong to their respective owners.

1 Background

Converters from Training Frameworks to NNEF, and from NNEF to Training Frameworks, are an essential part of NNEF success as an exchange format for neural networks.

The NNEF working group focuses on converters to and from a handful of selected frameworks. Conversion to NNEF is more important initially, in order to allow HW vendors to write their own converters from NNEF to the vendor specific distribution format. However, converters are easier to design and implement with both directions in mind. Therefore we describe this project with both directions in mind, putting more emphasis on the framework to NNEF direction.

The goal of this project is to procure an implementation of a converter between TensorFlow and NNEF. This converter will be uploaded by Khronos to GitHub.

2 Requirements

2.1 General

The project will deliver a converter between Tensorflow and NNEF that receives a TensorFlow protobuf file and generates semantically and functionally equivalent NNEF container, and is able to convert the NNEF container back to a TensorFlow protobuf file which when executed in TensorFlow produces equivalent results with the original source of conversion (although the backward conversion may not result in a one equivalent to the original protobuf).

2.2 Implementation

The converter will be written in Python, with minimal dependency on 3rd party components (either as source or as binary).

The implementer is required to provide the list of 3rd party components believed will be needed to implement the converter as part of the SOW definition. This list should include only components which are open source, preferably under Apache 2.0 license, and should get Khronos approval as part of accepting the work. Any additional component found during the work on the converter should be communicated to Khronos and get approved as well as a condition for accepting the resulted converter.

The Tensorflow library that parses the protobuf [3] is considered approved as it is required. Furthermore, the NNEF parser open-sourced by the NNEF group [5] is also considered approved and required.

2.3 Input of Converter to NNEF

A trained model can be distributed in various ways with TensorFlow: session checkpoint (multiple files created with tf.train.Saver), python network creation function with additional weight data file (for example slim tensorflow models), frozen model.

A frozen model is a single binary protocol buffer file that contains both the graph and the weights. This is a serialized representation of a GraphDef object in which networks weights are constant. The converter takes frozen

models as input since it is well suited for deploying a trained network into a production environment. Frozen models can be created with the TensorFlow *freeze_graph* tool.

The converter's input will then be a frozen tensorflow model file, and a few optional parameters.

Required:

- A 'frozen' protobuf file

Optional:

- Output path for creating the textual file and subfolders tree. If not provided, outputs will be written to current directory.
- Generate a compressed zip container. If provided, compress the textual file and subfolders into a zip file.

2.4 Output of Converter to NNEF

The converter will generate a textual file and tree of subfolders conforming to the container organization defined in NNEF Chapter 5.

In addition, the Converter will output a conversion report log, describing the number of operations successfully converted, warnings on possible conversion issues, and errors encountered during the conversion.

2.5 Conversion Requirements

The converter may optionally receive a list of external inputs and/or outputs. The converter is required to analyze the GraphDef object description which appears in the network definition protobuf (see Ref [3]) file in the following manner:

1. Inputs
    1.1. If the converter received a list of external inputs as a parameter, it shall use them as the list of graph inputs for this conversion operation. Any other inputs in the provided GraphDef object shall be discarded, and corresponding connected graph operations shall not be converted.
    1.2. If list of inputs is not provided, the converter is required to analyze the graph expressed by the GraphDef object, find all graph inputs, and write them into the graph in the converted NNEF textual file.
2. Outputs
    2.1. If the converter received a list of external outputs as a parameter, it shall use them as the list of graph outputs for this conversion operation. Any other outputs in the provided GraphDef object shall be discarded, and corresponding connected graph operations that generate this discarded outputs shall not be converted.
    2.2. If list of outputs is not provided, the converter is required to analyze the graph expressed by the GraphDef object, find all graph outputs, and write them into the graph in the converted NNEF textual file.
3. Operations
    3.1. For each node (operation) in the GraphDef, the converter is required to extract the operation type and identify if it is supported by NNEF. A minimal list of supported types for conversion is provided in section 2.6.
    3.2. If supported, the converter will create an NNEF equivalent description in the textual file of the operation type, inputs, outputs, arguments and any additional field required to create the description.
    3.3. The converter will correctly map tensor connections between operations outputs to other operations inputs, ensuring graph-level uniqueness of the tensor names.

3.4. The converter will correctly handle connection of external tensors such as weights, biases, etc, ensuring correct name mapping to relevant tensor folder and file name.

2.6 Backward Conversion

The conversion from NNEF to TensorFlow will be implemented as a separate tool (Python script). This tool may receive any NNEF file. For NNEF files that are the result of a conversion from TensorFlow to NNEF, this tool must successfully convert the network back to TensorFlow. For other NNEF files, the conversion may fail if it encounters features of NNEF that are unsupported by TensorFlow (certain operations or parameters of operations). The backward conversion process is otherwise similar to the conversion to NNEF described above, with the simplification that no handling of externally provided list of inputs and/or outputs need to be supported, only what is described by the NNEF file itself.

2.7 Required Operations

The table below provides the minimal list of operations required to be supported by the converter.

Tensorflow Operation	Suggested NNEF primitive	Notes/Restrictions
tf.get_variable, tf.Variable	variable	
tf.placeholder	external	
tf.constant	constant	
tf.concat	concat	
tf.split	split	
tf.reshape	reshape	
tf.transpose	transpose	
tf.add	add	
tf.subtract	sub	
tf.mul	mul	
tf.div	div	
tf.pow	pow	
tf.logical_and	and	
tf.logical_or	or	
tf.logical_not	not	
tf.negative	neg	

tf.abs	abs	
tf.sign	sign	
tf.exp	exp	
tf.log	log	
tf.sqrt	sqrt	
tf.square	sqr	
tf.floor	floor	
tf.ceil	ceil	
tf.round	round	
tf.greater	gt	
tf.greater_equal	ge	
tf.less	lt	
tf.less_equal	le	
tf.equal	eq	
tf.not_equal	ne	
tf.minimum	min	
tf.maximum	max	
tf.where	select	
tf.assign	update	
tf.matmul	matmul	
tf.reduce_sum	sum_reduce	
tf.reduce_mean	mean_reduce	
tf.reduce_max	max_reduce	
tf.sigmoid, tf.nn.sigmoid	sigmoid	
tf.tanh, tf.nn.tanh	tanh	

tf.nn.elu	elu	
tf.nn.relu	relu	
tf.nn.softsign	softsign	
tf.nn.softplus	softplus	
tf.nn.softmax	softmax	
tf.nn.conv1d, tf.nn.conv2d, tf.nn.conv3d, tf.nn.convolution	conv	
tf.nn.conv2d_transpose, tf.nn.conv3d_transpose	deconv	
tf.nn.depthwise_conv2d	planewise_conv	
tf.nn.separable_conv2d	conv	
tf.nn.max_pool	max_pool	
tf.nn.max_pool_with_argmax	max_pool_with_index	
tf.nn.avg_pool	avg_pool	
tf.nn.lrn, tf.nn.local_response_normalization	local_response_normalization	
tf.nn.batch_normalization	batch_normalization	This is broken into a subgraph by TF. Optionally, generating a single NNEF node requires some pattern matching
tf.nn.fused_batch_norm	batch_normalization	
tf.nn.l2_normalize	l2_normalization	
tf.nn.bias_add	add	
tf.image.resize_images	multilinear_upsample / nearest_upsample / nearest_downsample / area_downsample	

tf.pad		Must be merged into the following sliding window op

Notes:

- Some operations may require two-pass conversion. NNEF members with experience of conversion will be available for more information.
- Some operations are broken down into more primitive operations by TensorFlow. However, the scoping mechanism of TensorFlow preserves higher level information about these groups of operations, which may be used to fuse those operations back to higher level ones. Such a functionality is not required but may be performed by the exporter.
- Some operations may be broken up into primitives by TensorFlow (as a custom implementation), which are not supported by NNEF, but the whole higher level operation is supported (for example dilated convolution). In this case, the higher level operation is the only way to export to NNEF.

3 Deliverables and Acceptance Criteria

3.1 Deliveries

The scope of the Tensorflow to NNEF converter Implementation project will include the following deliverables:

- All source code for the converter
- Implementation notes document summarizing implementation decisions made during the course of the project
- A set of simplified Tensorflow models which test the operations defined in section 2.6.
- A detailed log of running the converter on the Tensorflow model zoo models defined in 3.3, and the simplified Tensorflow models described on previous bullet.

3.2 Review Period

1. The working group will have a review period of 4 weeks to review the converter code. At the end, it will provide a list of issues to be fixed, ranging these issues as critical/high/medium/low.
2. All critical and high issues need to be fixed/addressed as part of the Acceptance Criteria.

3.3 Acceptance Criteria

1. The Tensorflow to NNEF converter is required to be able to successfully convert the following Tensorflow models which appear on the Model Zoo (see Ref [2]).
2. The Tensorflow to NNEF converter is required to convert a set of simplified Tensorflow models which cover all operations described in section 2.6.
3. All converted models from #1 and #2, provided as NNEF containers, should successfully pass the NNEF validator check.

4. All converted models are required to be possible to convert back to TensorFlow from NNEF, and the result of the backward conversion should result in networks functionally equivalent to the ones from which the conversion to NNEF was started.
5. All issues found during the review period and classified as critical/high should have been fixed.

4 Project Scoping and Schedule

The NNEF working group estimates that the project can achieve complete implementation, testing and documentation, in no more than 6 man weeks.

Below are the suggested project milestones. We will assess progress on a weekly basis, so the feature coverage timeline below is only a rough guideline to the order in which we expect to have validator and tests written. Please provide detailed milestone dates that you can commit to delivering:

Milestone	Date	Content	Notes
M1		Khronos releases RFQ	
M2	M1 + 4 weeks	RFQ responses received by Khronos	
M3	M2 + 2 weeks	Contractor selected and notified	
M4	M3 + 3 weeks	Contract executed and start of work	10% of money is provided
M5	M4 + 3 weeks	Simple network end to end functional with minimum operations implemented.	30% of money is provided
M6	M5 + 3 weeks	100% of converter implemented	30% of money is provided
M7	M6 + 4 weeks	Review Period over	
M8	M7 + 3 weeks	Issues found during review period fixed, project is complete.	30% of money is provided

5 Khronos NDA, Contractor and Membership Agreement

The selected contractor will be required to execute the standard Khronos Contractors Agreement with milestones and costs entered into Exhibit B and Contractor Disclosures entered into Exhibit C.

If the selected contractor is not a Khronos member, the contractor shall also be required to execute the standard Khronos membership agreement (with fees waived) for the duration of the project in order to gain access to confidential materials and meetings for the sole purpose completing deliverables in this RFQ.

No work shall begin, and Khronos shall be liable for no costs or expenses, until the selected contractor is in receipt of an executed contractor's agreement.

It is important that contractors understand that Khronos will be assessing progress on a regular basis, and reserve the right to terminate or renegotiate the contract in the event insufficient progress is being made.

6 RFQ Responses

The RFQ response materials will form the basis for detailed milestone and cost negotiations for the final contract with the selected vendor or vendors. Please provide the following information in the format of your choice:

- Identification of deliverables on which you wish to bid;
- Proposed schedule, highlighting any differences from the suggested milestones in Section 4;
- The hourly cost for engineering resources from your company, the minimum and maximum number of hours you can commit to this project on a weekly basis, and a description of the qualification of the engineering resource(s) which would be used;
- The total project cost to Khronos. We can accept time and material or fixed cost bids – but strongly prefer fixed cost proposals;
- An indication you are willing to work under the terms of the standard Khronos Contractor Agreement and execute the Khronos membership agreement if necessary;
- Any particular issues or risk factors that you wish to highlight;
- Supporting materials, including background materials about your company, highlighting other relevant experience and expertise for this project.

RFQ responses are requested by the close of business on TBD and should be sent to nnef-rfq@khronos.org.

7 References

[1] TensorFlow project on GitHub : https://github.com/tensorflow/tensorflow

[2] TensorFlow Model Zoo : https://github.com/tensorflow/models/tree/master/research/slim, Pretrained models section

[3] TensorFlow Protobuf description: https://www.tensorflow.org/extend/tool_developers/

[4] TensorFlow API docs : https://www.tensorflow.org/api_docs/

[5] NNEF parser : https://github.com/KhronosGroup/NNEF-Tools