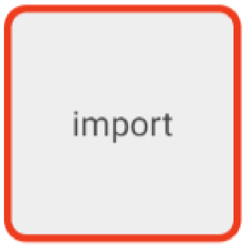
Tensorflow Model Basics

Inspecting the model using tensorboard

After using `import_pb.py` to generate the tensorboard files for the model, we can launch tensorboard to inspect the model in the web browser.

After loading the model, we only see a single block called “import” on the first look.



Double clicking the block will allow us to inspect the inside of a block.

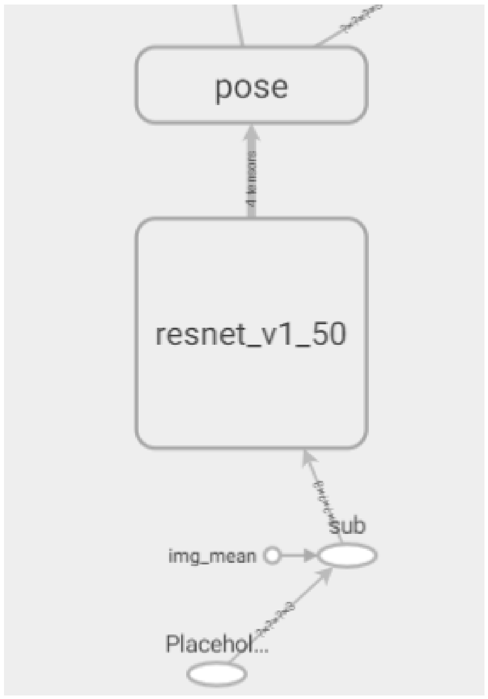
Top-level structure

If we unravel the top level of the block, we can see that it contains several parts (from bottom to up, or input to output):

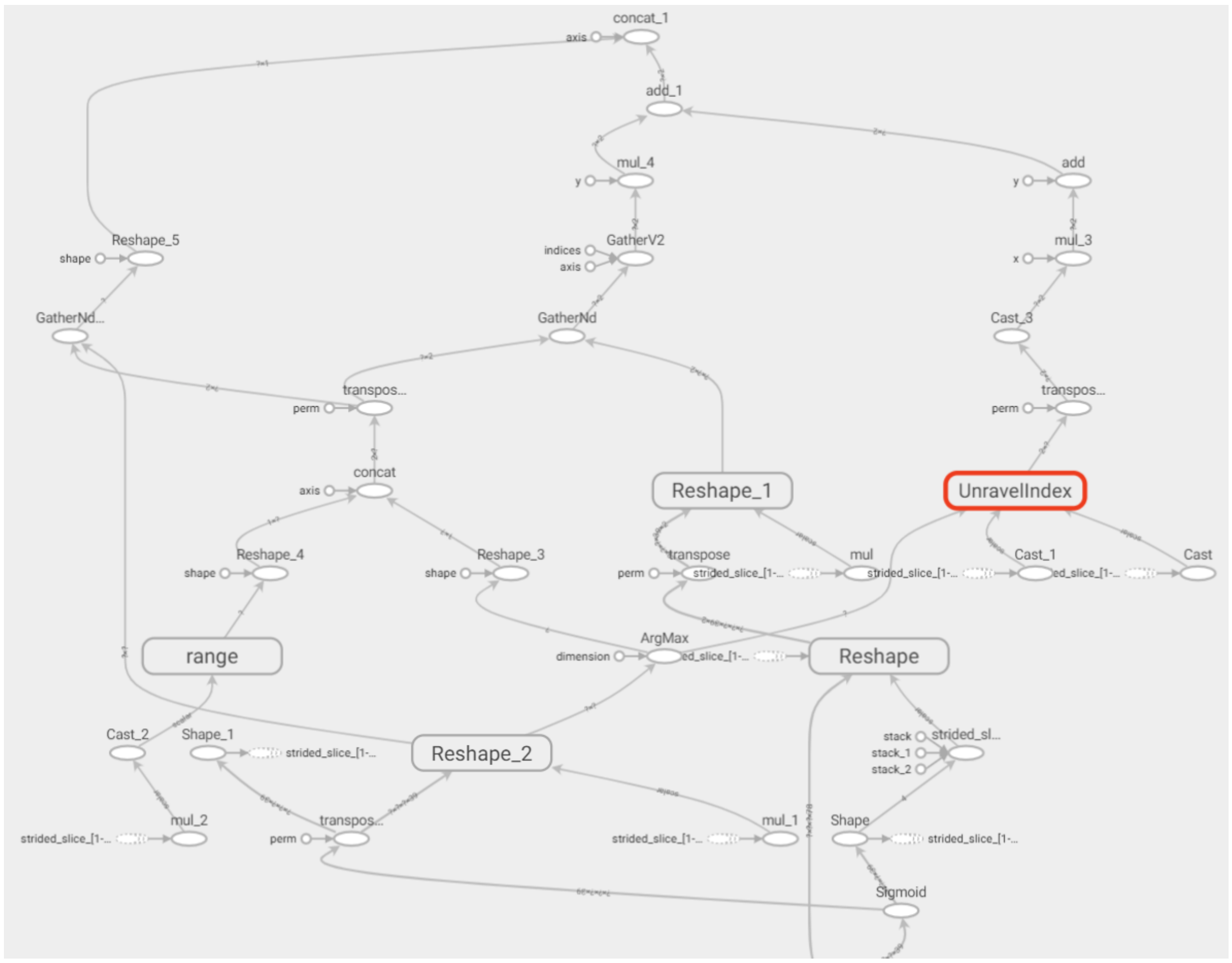
- Input and preprocessing: the bottom “placeholder” and “sub” operation
- “resnet_v1_50” block
- “pose” block
- The rest (very big structure of custom operations)

The following picture shows the parts

First three parts



The last part



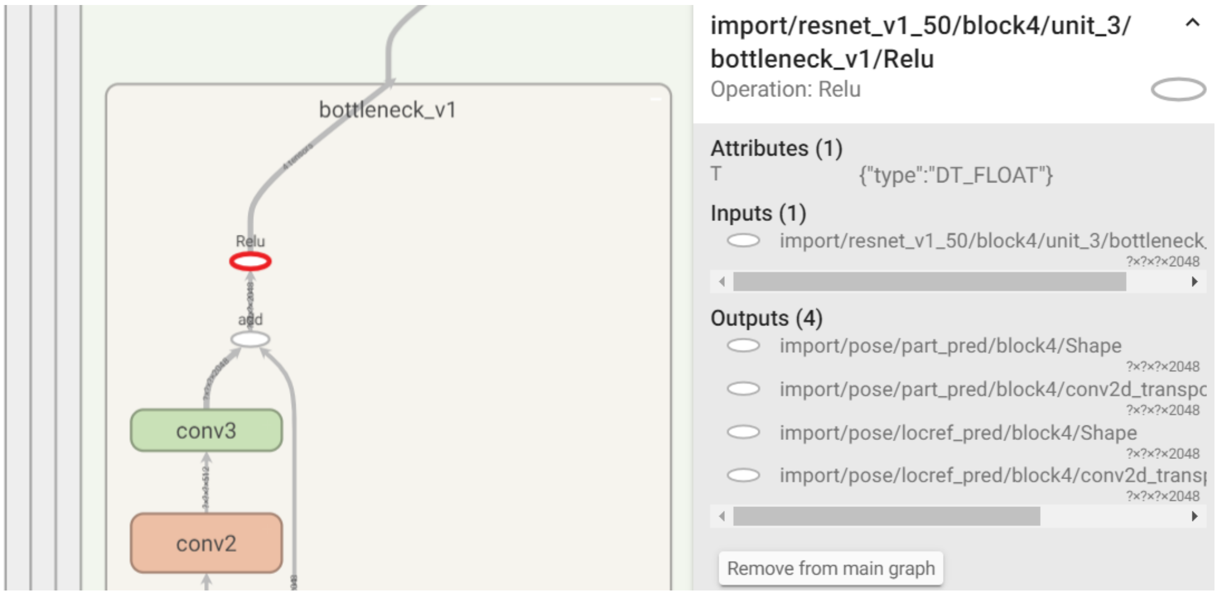
The “resnet” and “pose” blocks include more low-level structures and can be further revealed by double clicking on the blocks.

It turns out that the first three part are mostly traditional neural-network operations and is readily portable to Coral accelerator. But the last part contains operations that are not suitable to run on TPU. The operations are Reshape and UnravelIndex (the red operation).

Inspecting Network Detail

To “split” the model, we need to get the split point, which defined by a series of “operations” (the lowest level node in the graph) and the associated tensors (the input/output data of the operation).

For example, if we unwrap the resnet block and go to the deepest level, we can find the last operation to be a Relu node (the red circle).



import/resnet_v1_50/block4/unit_3/
bottleneck_v1/Relu
Operation: Relu

Attributes (1)

T	{"type":"DT_FLOAT"}

Inputs (1)

import/resnet_v1_50/block4/unit_3/bottleneck
?×?×?×2048

Outputs (4)

import/pose/part_pred/block4/Shape
?×?×?×2048
import/pose/part_pred/block4/conv2d_transpc
?×?×?×2048
import/pose/locref_pred/block4/Shape
?×?×?×2048
import/pose/locref_pred/block4/conv2d_trans
?×?×?×2048

Remove from main graph

The panel on the right reveals the name and dimensions of input and output tensors. The names will be used in the splitting.

Manually split a network

Step 1: Loading a model

```
import tensorflow as tf

tfv1 = tf.compat.v1
gdef = tfv1.GraphDef()
with tfv1.io.gfile.GFile("DLC_ma_superquadruped_resnet_50_iteration-0_shu
ffle-1/snapshot-700000.pb","rb") as f:
  gdef.ParseFromString(f.read())

g = tf.Graph()
with g.as_default():
  tf.graph_util.import_graph_def(gdef, name='DLC')
```

Step 2: Create subgraph

Here, if we decide that we want to split the network into two parts: the first part contains the first three blocks (input, resnet and pose), and the second part contains the rest. We need to inspect the operations at the output on the boundary.

The “pose” block output two tensors from two subblocks, and the names of the operations are called `pose/part_pred/block4/BiasAdd` and `pose/locref_pred/block4/BiasAdd`.

The following script demonstrates the process of splitting the graph into two parts

```
import tensorflow as tf
tfv1 = tf.compat.v1
gdef = tfv1.GraphDef()
with tfv1.io.gfile.GFile("DLC_ma_superquadruped_resnet_50_iteration-0_shu
ffle-1/snapshot-700000.pb","rb") as f:
  gdef.ParseFromString(f.read())

>>> g = tf.Graph()
>>> with g.as_default():
...   tf.graph_util.import_graph_def(gdef, name="DLC")
...
>>> input_tensor_name = str(g.get_operations()[0].name) + ":0"
>>> input_tensor = g.get_tensor_by_name(input_tensor_name)
>>> input_tensor
<tf.Tensor 'DLC/Placeholder:0' shape=(None, None, None, 3) dtype=float32>

>>> part_pred_output_name = 'DLC/pose/part_pred/block4/BiasAdd:0'
>>> part_pred_output_tensor = g.get_tensor_by_name(part_pred_output_name)
>>> locref_pred_output_name = 'DLC/pose/locref_pred/block4/BiasAdd:0'
>>> locref_pred_output_name = g.get_tensor_by_name(locref_pred_output_nam
e)
>>> with tfv1.Session(graph=g) as s:
...   tfv1.saved_model.simple_save(session=s, export_dir='DLC_ma_p1/', in
puts={'input':input_tensor}, outputs={'part_pred_output': part_pred_outpu
t_tensor, 'locref_pred_output': locref_pred_output_name})
...
WARNING:tensorflow:From <stdin>:2: simple_save (from tensorflow.python.sa
ved_model.simple_save) is deprecated and will be removed in a future vers
ion.
Instructions for updating:
This function will only be available through the v1 compatibility library
as tf.compat.v1.saved_model.simple_save.
WARNING:tensorflow:From /home/yunze/miniforge3/envs/dlc-live/lib/python3.
7/site-packages/tensorflow/python/saved_model/signature_def_utils_impl.p
y:204: build_tensor_info (from tensorflow.python.saved_model.utils_impl)
is deprecated and will be removed in a future version.
```

```
Instructions for updating:
This function will only be available through the v1 compatibility library
as tf.compat.v1.saved_model.utils.build_tensor_info or tf.compat.v1.saved
_model.build_tensor_info.
INFO:tensorflow:Assets added to graph.
INFO:tensorflow:No assets to write.
INFO:tensorflow:SavedModel written to: DLC_ma_p1/saved_model.pb

>>> gdef_sub = tfv1.graph_util.extract_sub_graph(gdef, ['pose/part_pred/b
lock4/BiasAdd', 'pose/locref_pred/block4/BiasAdd'])

>>> with g2.as_default():
...   tf.graph_util.import_graph_def(gdef_sub, name='')
...
>>> g2.get_operations()[0]
<tf.Operation 'Placeholder' type=Placeholder>
>>> g2.get_operations()[-1]
<tf.Operation 'pose/locref_pred/block4/BiasAdd' type=BiasAdd>
>>> g2.get_operations()[-2]
<tf.Operation 'pose/locref_pred/block4/BiasAdd/ReadVariableOp' type=Ident
ity>
>>> g2_input=g2.get_tensor_by_name('Placeholder:0')
>>> g2_part_pred_output=g2.get_tensor_by_name('pose/part_pred/block4/Bias
Add:0')
>>> g2_locref_pred_output=g2.get_tensor_by_name('pose/locref_pred/block4/
BiasAdd:0')
>>> with tfv1.Session(graph=g2) as s2:
...   tfv1.saved_model.simple_save(session=s2, export_dir='DLC_ma_sub_p
1/', inputs={'input':g2_input}, outputs={'part_pred_output': g2_part_pred
_output, 'locref_pred_output': g2_locref_pred_output})
...
INFO:tensorflow:Assets added to graph.
INFO:tensorflow:No assets to write.
INFO:tensorflow:SavedModel written to: DLC_ma_sub_p1/saved_model.pb
>>> output_tensor = g.get_tensor_by_name('DLC/concat_1:0')
>>> with tfv1.Session(graph=g) as s:
...   tfv1.saved_model.simple_save(session=s, export_dir='DLC_ma_p2/', in
puts={'part_pred_input': part_pred_output_tensor, 'locref_pred_input': lo
cref_pred_output_name}, outputs={'output': output_tensor})
...
INFO:tensorflow:Assets added to graph.
```

```
INFO:tensorflow:No assets to write.
INFO:tensorflow:SavedModel written to: DLC_ma_p2/saved_model.pb
```

The two parts are saved into `DLC_ma_p1/` and `DLC_ma_p2/` directories.

Note: To convert the model for Coral to use, the model needs to be fixed in input size. The following code snippet fixes the input to 320×320 for the first part of the model.

(also refer to gen_lite_model.py)

```
tfv1 = tf.compat.v1
gdef = tfv1.GraphDef()
with tfv1.io.gfile.GFile("DLC_ma_superquadruped_resnet_50_iteration-0_shu
ffle-1/snapshot-700000.pb","rb") as f:
  gdef.ParseFromString(f.read())

g = tf.Graph()
with g.as_default():
  tf.graph_util.import_graph_def(gdef, name='')

>>> g.get_operations()[0]
<tf.Operation 'Placeholder' type=Placeholder>
>>> g.get_operations()[3]
<tf.Operation 'resnet_v1_50/Pad/paddings' type=Const>
>>> print(g.get_tensor_by_name('pose/part_pred/block4/BiasAdd:0'))
Tensor("pose/part_pred/block4/BiasAdd:0", shape=(None, None, None, 39), d
type=float32)
>>>
new_graph=tf.Graph()

with new_graph.as_default():
  new_input = tfv1.placeholder(dtype=tf.float32, shape=[1,320,320,3], nam
e='Placeholder')
  tfv1.import_graph_def(g.as_graph_def(), name='', input_map={'Placeholde
r':new_input})


>>> new_graph.get_operations()[0]
<tf.Operation 'Placeholder' type=Placeholder>
>>> new_graph.get_operations()[2]
<tf.Operation 'img_mean' type=Const>
```

```
>>> new_graph.get_operations()[3]
<tf.Operation 'sub' type=Sub>
>>> new_graph.get_operations()[1]
<tf.Operation 'Placeholder_1' type=Placeholder>
>>> print(new_graph.get_tensor_by_name('pose/part_pred/block4/BiasAdd:
0'))
Tensor("pose/part_pred/block4/BiasAdd:0", shape=(1, 40, 40, 39), dtype=fl
oat32)
```