ONNX-Tensorflow PR Creation Walk-Through

Winnie Tsang, Cognitive OpenTech

Data and AI
Open Source Dojo

Overview

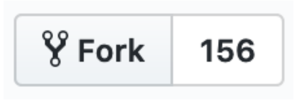
ONNX-Tensorflow is a converter that convert Open Neural Network Exchange (ONNX) model into Tensorflow model.

Git repository: https://github.com/onnx/onnx-tensorflow

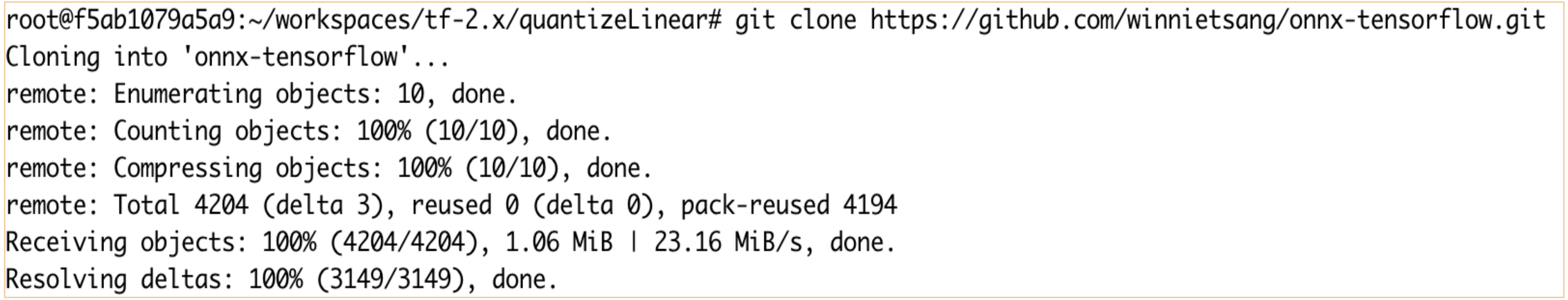
Issues: https://github.com/onnx/onnx-tensorflow/issues

Pull Requests: https://github.com/onnx/onnx-tensorflow/pulls

Setup Development Environment

- Create a fork of ONNX-Tensoflow in your git account by clicking 

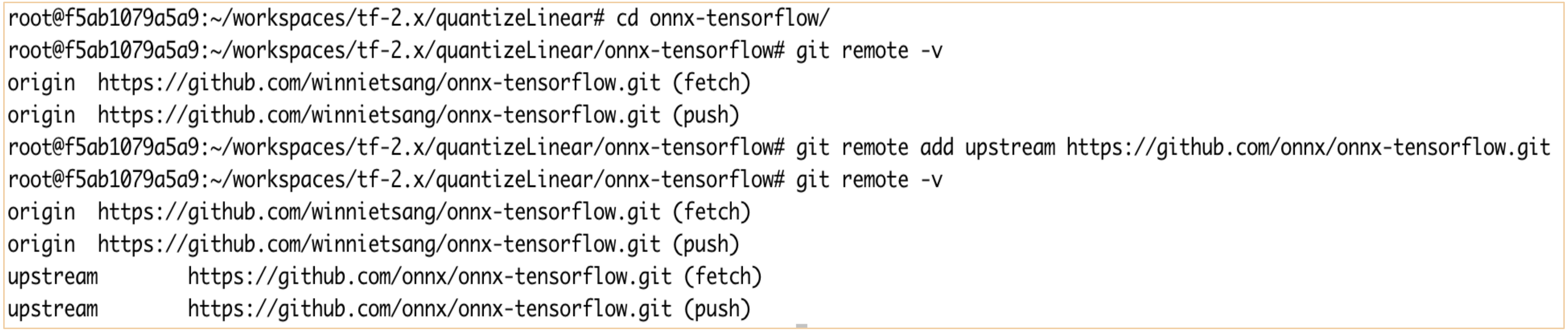
  button on the top right corner of ONNX-Tensorflow repository
- Clone your fork
  - *git clone https://github.com/<your-git-user-name>/onnx-tensorflow.git*



```
root@f5ab1079a5a9:~/workspaces/tf-2.x/quantizeLinear# git clone https://github.com/winnietsang/onnx-tensorflow.git
Cloning into 'onnx-tensorflow'...
remote: Enumerating objects: 10, done.
remote: Counting objects: 100% (10/10), done.
remote: Compressing objects: 100% (10/10), done.
remote: Total 4204 (delta 3), reused 0 (delta 0), pack-reused 4194
Receiving objects: 100% (4204/4204), 1.06 MiB | 23.16 MiB/s, done.
Resolving deltas: 100% (3149/3149), done.
```

Setup Development Environment (continue)

- Setup the upstream remote
  - *git remote add upstream https://github.com/onnx/onnx-tensorflow.git*



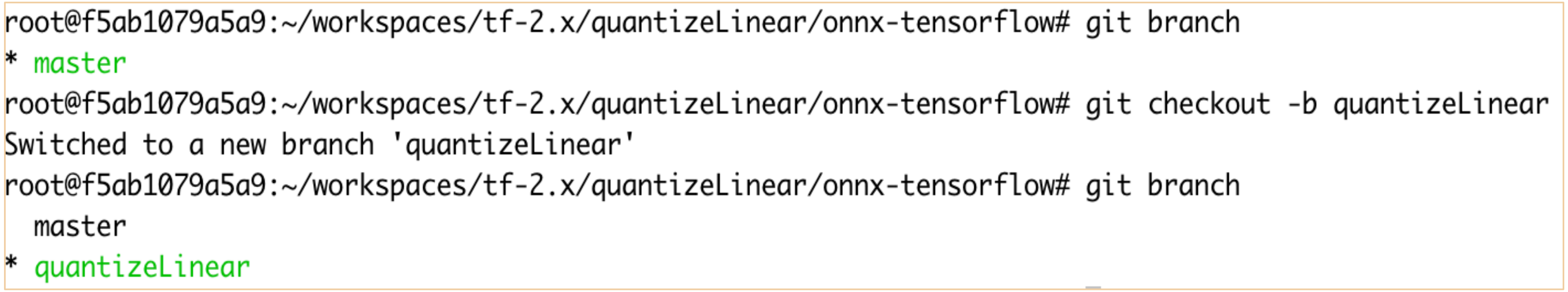
```
root@f5ab1079a5a9:~/workspaces/tf-2.x/quantizeLinear# cd onnx-tensorflow/
root@f5ab1079a5a9:~/workspaces/tf-2.x/quantizeLinear/onnx-tensorflow# git remote -v
origin  https://github.com/winnietsang/onnx-tensorflow.git (fetch)
origin  https://github.com/winnietsang/onnx-tensorflow.git (push)
root@f5ab1079a5a9:~/workspaces/tf-2.x/quantizeLinear/onnx-tensorflow# git remote add upstream https://github.com/onnx/onnx-tensorflow.git
root@f5ab1079a5a9:~/workspaces/tf-2.x/quantizeLinear/onnx-tensorflow# git remote -v
origin  https://github.com/winnietsang/onnx-tensorflow.git (fetch)
origin  https://github.com/winnietsang/onnx-tensorflow.git (push)
upstream        https://github.com/onnx/onnx-tensorflow.git (fetch)
upstream        https://github.com/onnx/onnx-tensorflow.git (push)
```

Setup Development Environment (continue)

- Verify your fork master is up-to-date by navigate your browser to your fork https://github.com/<your-git-user-name>/onnx-tensorflow
  - If your master is up-to-date then you should find “This branch is even with onnx:master” display on the top.
  - If you don’t see the above message then your master is out-of-date, please run the following commands to update it
    - *git fetch upstream*
    - *git merge upstream/master*
    - *git push origin master*

Setup Development Environment (continue)

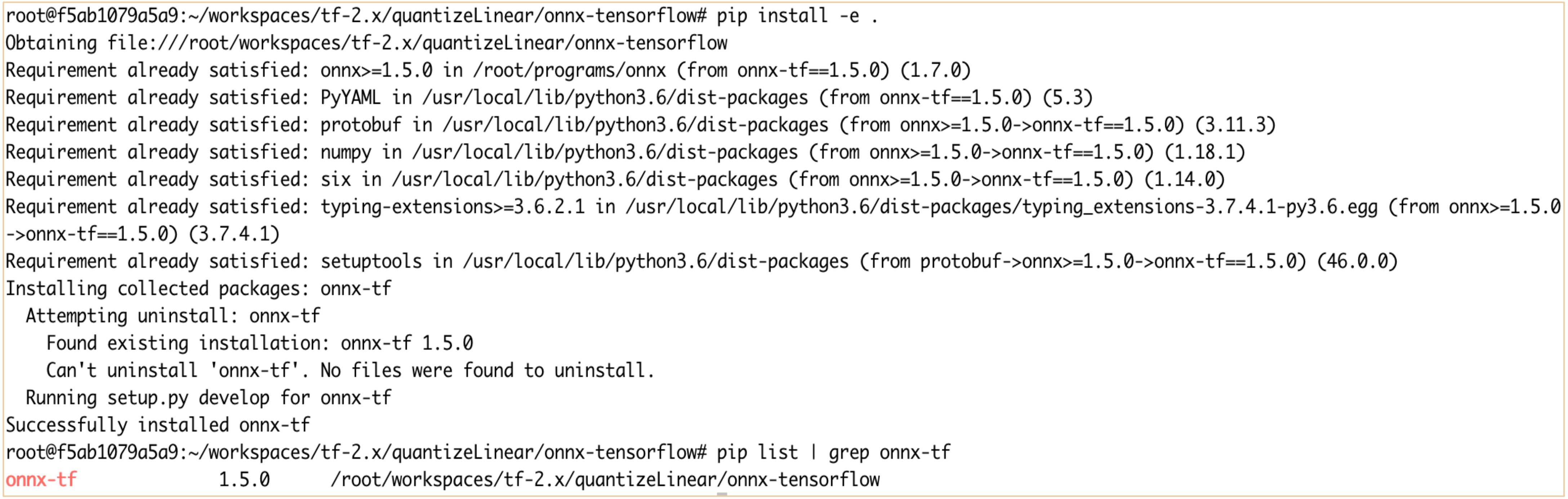
- Create a branch under your fork as your development environment
  - *git checkout -b <your-branch-name>*



```
root@f5ab1079a5a9:~/workspaces/tf-2.x/quantizeLinear/onnx-tensorflow# git branch
* master
root@f5ab1079a5a9:~/workspaces/tf-2.x/quantizeLinear/onnx-tensorflow# git checkout -b quantizeLinear
Switched to a new branch 'quantizeLinear'
root@f5ab1079a5a9:~/workspaces/tf-2.x/quantizeLinear/onnx-tensorflow# git branch
  master
* quantizeLinear
```

Setup Development Environment (continue)

- Use your branch to build onnx-tf
  - *pip install -e .*



```
root@f5ab1079a5a9:~/workspaces/tf-2.x/quantizeLinear/onnx-tensorflow# pip install -e .
Obtaining file:///root/workspaces/tf-2.x/quantizeLinear/onnx-tensorflow
Requirement already satisfied: onnx>=1.5.0 in /root/programs/onnx (from onnx-tf==1.5.0) (1.7.0)
Requirement already satisfied: PyYAML in /usr/local/lib/python3.6/dist-packages (from onnx-tf==1.5.0) (5.3)
Requirement already satisfied: protobuf in /usr/local/lib/python3.6/dist-packages (from onnx>=1.5.0->onnx-tf==1.5.0) (3.11.3)
Requirement already satisfied: numpy in /usr/local/lib/python3.6/dist-packages (from onnx>=1.5.0->onnx-tf==1.5.0) (1.18.1)
Requirement already satisfied: six in /usr/local/lib/python3.6/dist-packages (from onnx>=1.5.0->onnx-tf==1.5.0) (1.14.0)
Requirement already satisfied: typing-extensions>=3.6.2.1 in /usr/local/lib/python3.6/dist-packages/typing_extensions-3.7.4.1-py3.6.egg (from onnx>=1.5.0
->onnx-tf==1.5.0) (3.7.4.1)
Requirement already satisfied: setuptools in /usr/local/lib/python3.6/dist-packages (from protobuf->onnx>=1.5.0->onnx-tf==1.5.0) (46.0.0)
Installing collected packages: onnx-tf
  Attempting uninstall: onnx-tf
    Found existing installation: onnx-tf 1.5.0
    Can't uninstall 'onnx-tf'. No files were found to uninstall.
  Running setup.py develop for onnx-tf
Successfully installed onnx-tf
root@f5ab1079a5a9:~/workspaces/tf-2.x/quantizeLinear/onnx-tensorflow# pip list | grep onnx-tf
onnx-tf                1.5.0      /root/workspaces/tf-2.x/quantizeLinear/onnx-tensorflow
```

ONNX-Tensorflow Potential Pull Request (PR)

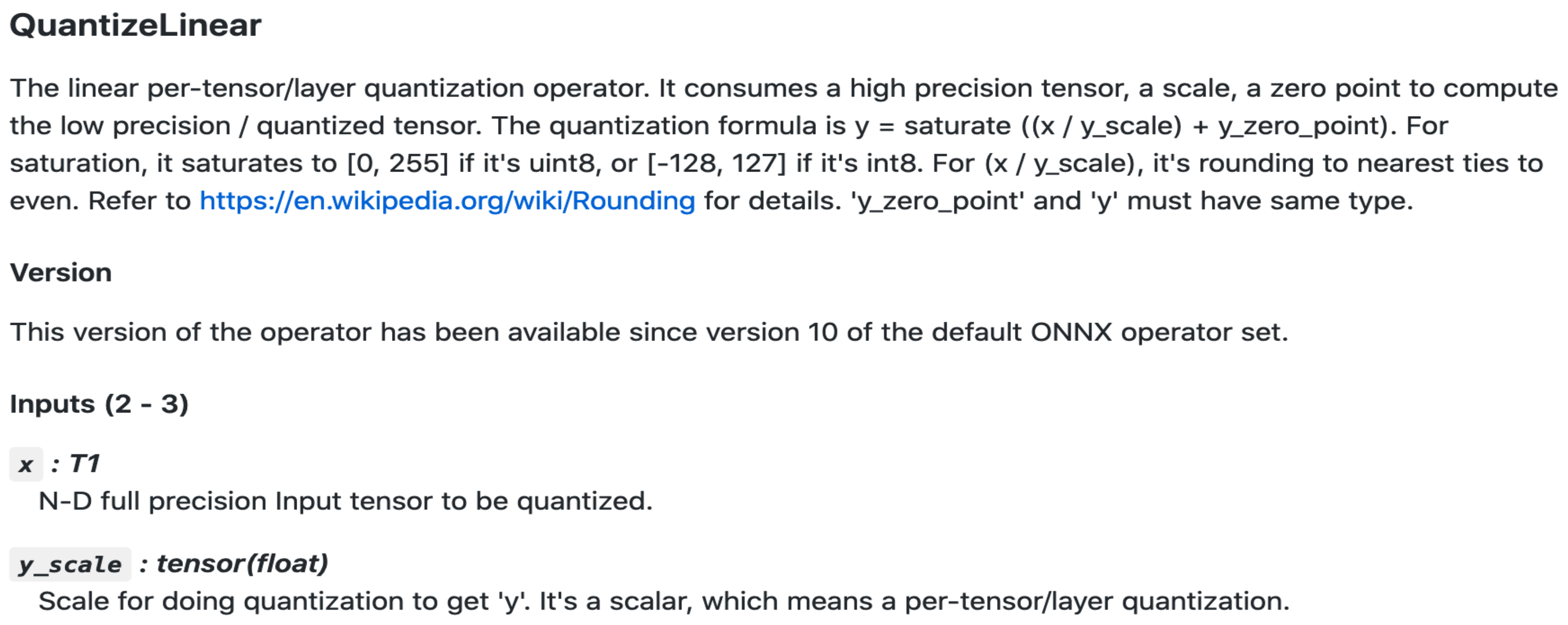
1. Fix an existing issue in https://github.com/onnx/onnx-tensorflow/issues
   - Claim the issue by post a comment in that issue about your intension to work on it.
2. Identify a new issue and fix it
   - File the issue and claim it
3. Add support for a new OpSet version of an existing ONNX operator
   - File the issue and claim it
4. Add support for a new ONNX operator
   - File the issue and claim it

Process to Create a Handler for a New ONNX Operator

1. Identify and study the specification of the new ONNX operator from https://github.com/onnx/onnx/blob/master/docs/Operators.md
2. Identify the potential Tensorflow operator(s) that can perform the equivalent function from https://www.tensorflow.org/api_docs/python/tf
3. Add a test case for this operator in https://github.com/onnx/onnx-tensorflow/blob/master/test/backend/test_node.py
4. Create the new handler file under https://github.com/onnx/onnx-tensorflow/tree/master/onnx_tf/handlers/backend
5. Develop the handler and test it by running the test case in step 3

Create QuantizeLinear Handler Process Walk-Through

1. Study the specification of QuantizeLinear in ONNX operators doc
   https://github.com/onnx/onnx/blob/master/docs/Operators.md#quantizelinear



QuantizeLinear

The linear per-tensor/layer quantization operator. It consumes a high precision tensor, a scale, a zero point to compute the low precision / quantized tensor. The quantization formula is y = saturate ((x / y_scale) + y_zero_point). For saturation, it saturates to [0, 255] if it's uint8, or [-128, 127] if it's int8. For (x / y_scale), it's rounding to nearest ties to even. Refer to https://en.wikipedia.org/wiki/Rounding for details. 'y_zero_point' and 'y' must have same type.

Version

This version of the operator has been available since version 10 of the default ONNX operator set.

Inputs (2 - 3)

`x` : *T1*
N-D full precision Input tensor to be quantized.

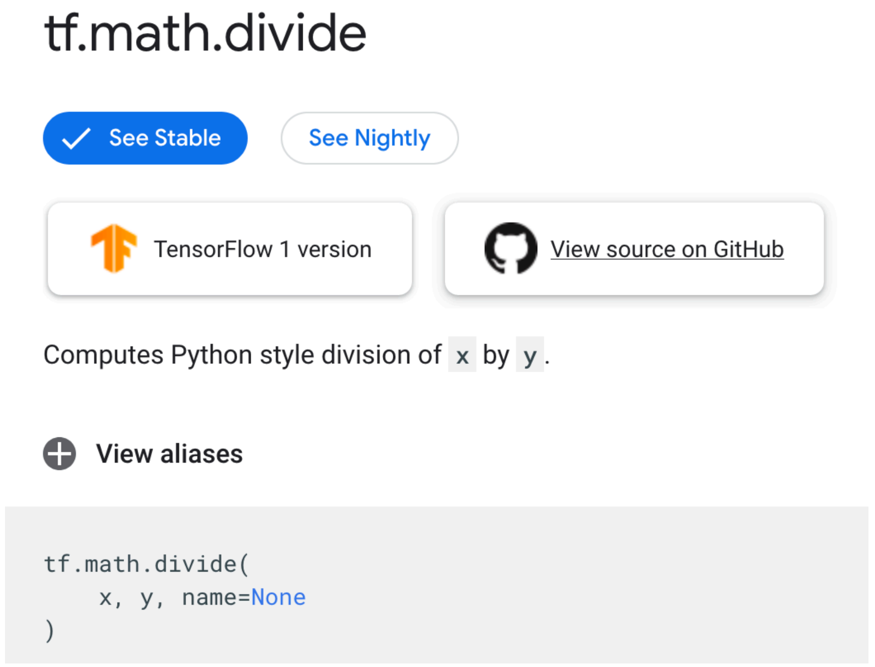
`y_scale` : *tensor(float)*
Scale for doing quantization to get 'y'. It's a scalar, which means a per-tensor/layer quantization.

ONNX Specification of QuantizeLinear Operator

- The quantization formula is y = saturate ((x / y_scale) + y_zero_point).
- For saturation, it saturates to [0, 255] if it's uint8, or [-128, 127] if it's int8.
- For (x / y_scale), it's rounding to nearest ties to even.
- 'y_zero_point' and 'y' must have same type.
- ‘y_zero_point’ default value is uint8 typed 0.
- Since version = 10
- Inputs (2-3)
    - x : tensor(float), tensor(int32)
    - y_scale : tensor(float)
    - y_zero_point (optional) : tensor(int8), tensor(uint8)
- Output: y : tensor(int8), tensor(uint8)

Create QuantizeLinear Handler Process Walk-Through (continue)

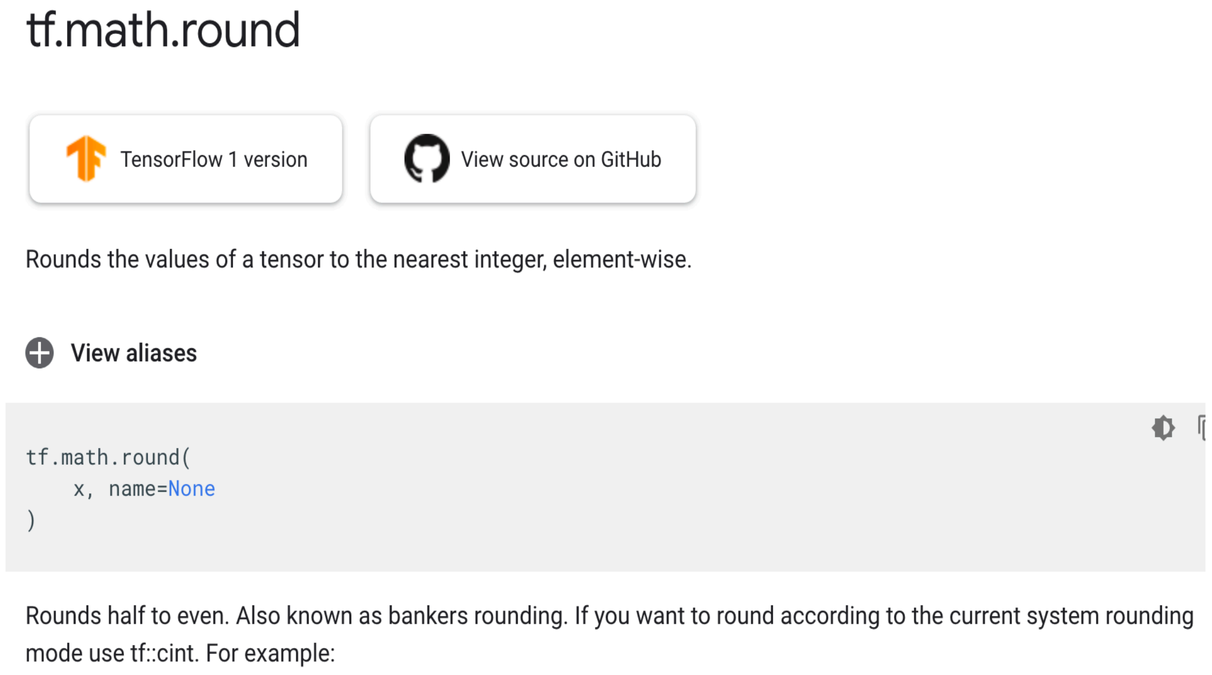
2. Identify the potential Tensorflow operator(s) that can calculate output "y" base on the quantization formula from https://www.tensorflow.org/api_docs/python/tf



tf.math.divide

Computes Python style division of `x` by `y`.

```
tf.math.divide(
    x, y, name=None
)
```



tf.math.round

Rounds the values of a tensor to the nearest integer, element-wise.

```
tf.math.round(
    x, name=None
)
```

Rounds half to even. Also known as bankers rounding. If you want to round according to the current system rounding mode use tf::cint. For example:

Tensorflow Operators to Perform ONNX QuantizeLinear Operator

- In Tensorflow, there is no Quantize Linear operator, so it need to use a set of Tensorflow operators to perform the equivalent function.
- x / y_scale can be performed by tf.divide
    - https://www.tensorflow.org/api_docs/python/tf/math/divide
- rounding (x / y_scale) can be performed by tf.round
    - https://www.tensorflow.org/api_docs/python/tf/math/round
- (x / y_scale) + y_zero_point can be performed by tf.add
    - https://www.tensorflow.org/api_docs/python/tf/math/add
- saturate ((x / y_scale) + y_zero_point) can be performed by tf.saturate_cast
    - https://www.tensorflow.org/api_docs/python/tf/dtypes/saturate_cast

Create QuantizeLinear Handler Process Walk-Through (continue)

3. Add the quantize_linear unit test into https://github.com/onnx/onnx-tensorflow/blob/master/test/backend/test_node.py
   - Create inputs for quantize_linear onnx node
   - Use onnx.helper.make_node to create the quantize_linear onnx node
   - Use onnx_tf.backend.run_node to convert this quantize_linear onnx node to tensorflow nodes
   - Use numpy to calculate the quantization formula as the expected value
   - Compare the result from run_node with the expected value

IBM

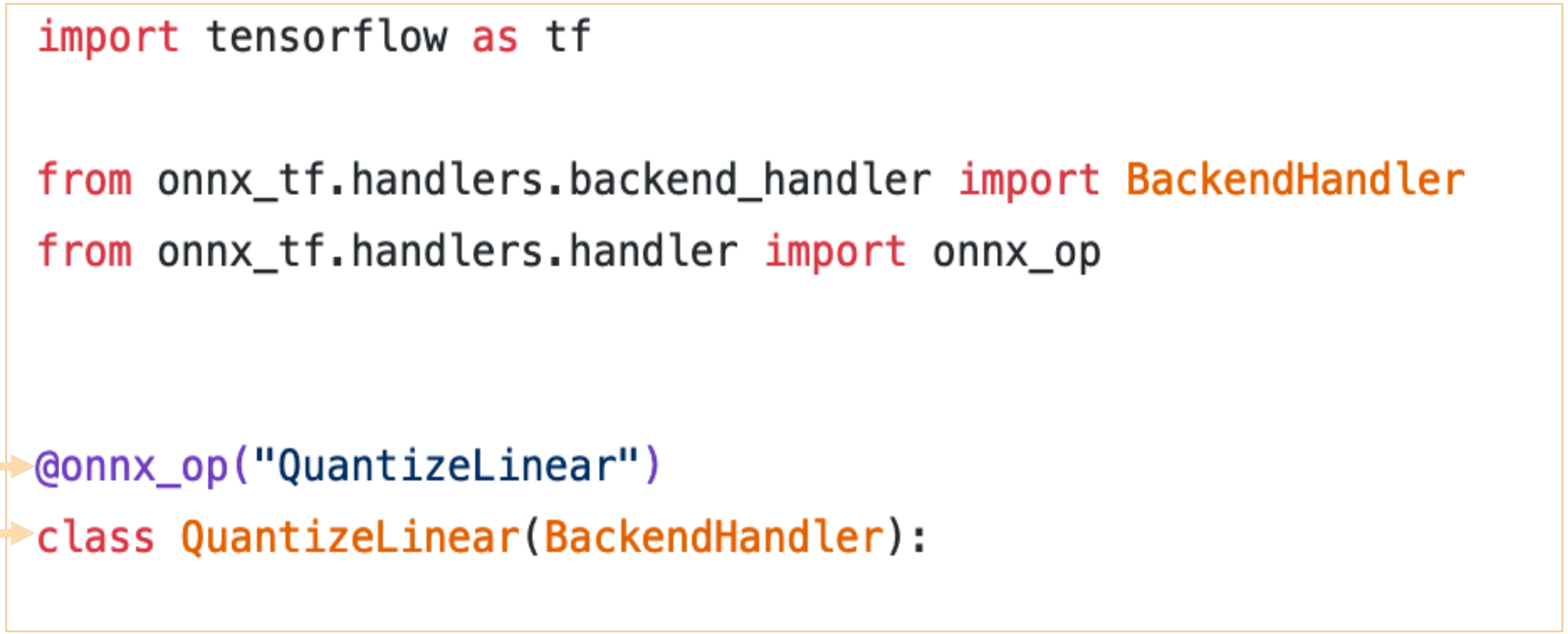
QuantizeLinear Unit Test in test_node.py

```python
def test_quantize_linear(self):
  node_def = helper.make_node("QuantizeLinear",
                              ["x", "y_scale", "y_zero_point"], ["y"])
  for x in [
      self._get_rnd_float32(-512., 512., [2, 6]),
      self._get_rnd_int(-512, 512, [2, 6])
  ]:
    y_scale = self._get_rnd_float32(-10., 10.)
    for y_zero_point in [
        self._get_rnd_int(-128, 127, dtype=np.int8),
        self._get_rnd_int(0, 255, dtype=np.uint8)
    ]:
      y = np.divide(x, y_scale)
      y = np.round(y)
      y = np.add(y, y_zero_point)
      if y_zero_point.dtype.type is np.int8:
        y = np.clip(y, -128, 127).astype(np.int8)
      else:
        y = np.clip(y, 0, 255).astype(np.uint8)
      output = run_node(node_def, [x, y_scale, y_zero_point])
      np.testing.assert_almost_equal(output["y"], y)
```

https://github.com/onnx/onnx-tensorflow/blob/master/test/backend/test_node.py

Create QuantizeLinear Handler Process Walk-Through (continue)

4. Create quantize_linear handler file under https://github.com/onnx/onnx-tensorflow/tree/master/onnx_tf/handlers/backend



```python
import tensorflow as tf

from onnx_tf.handlers.backend_handler import BackendHandler
from onnx_tf.handlers.handler import onnx_op


@onnx_op("QuantizeLinear")
class QuantizeLinear(BackendHandler):
```

Handle QuantizeLinear ONNX operator

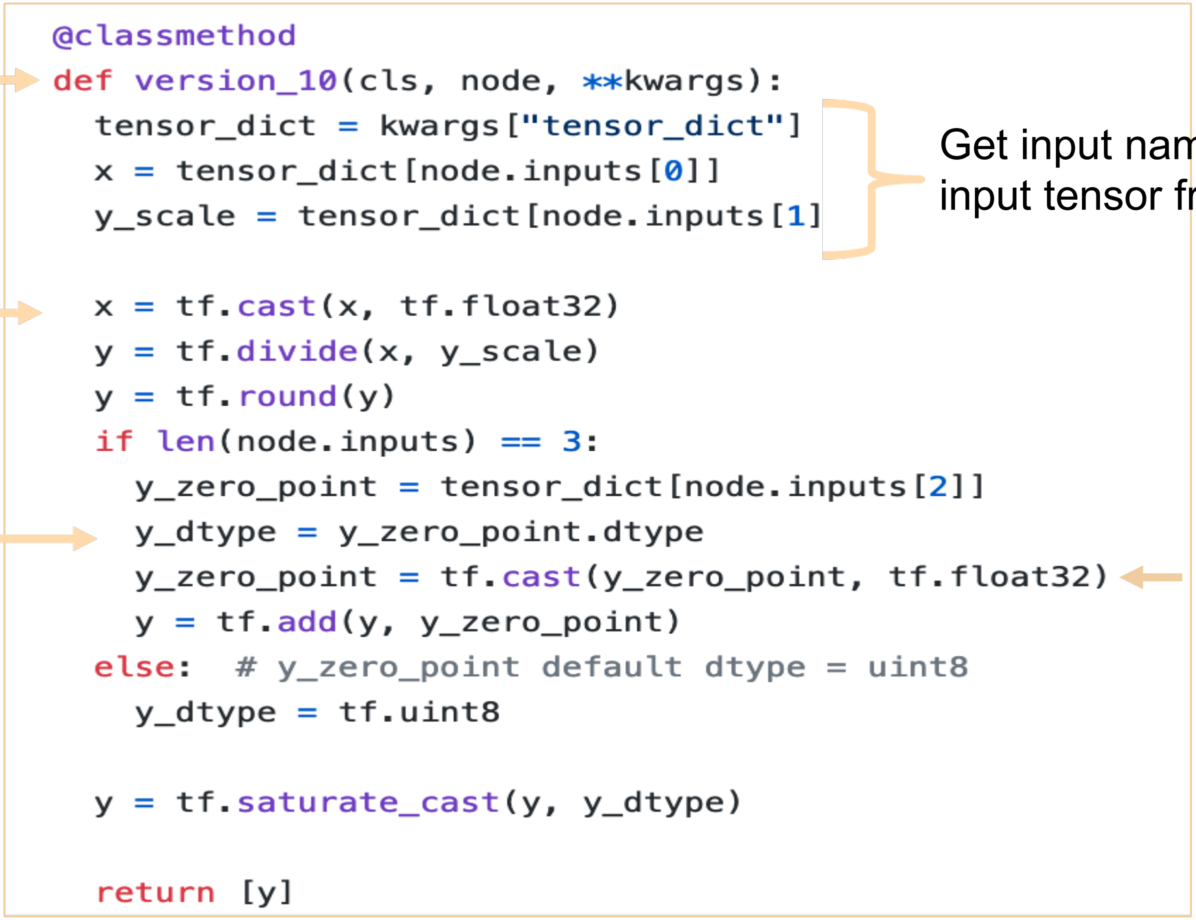
Inherit BackendHandler class

QuantizeLinear Handler

Define version_10 method to handle QuantizeLinear in opset 10 and above

Need to cast ‘x’ to the same dtype as ‘y_scale’ before division, because tf.divide required both inputs in the same dtype

‘y’ dtype need to be the same as ‘y_zero_point’ dtype



```
@classmethod
def version_10(cls, node, **kwargs):
  tensor_dict = kwargs["tensor_dict"]
  x = tensor_dict[node.inputs[0]]
  y_scale = tensor_dict[node.inputs[1]

  x = tf.cast(x, tf.float32)
  y = tf.divide(x, y_scale)
  y = tf.round(y)
  if len(node.inputs) == 3:
    y_zero_point = tensor_dict[node.inputs[2]]
    y_dtype = y_zero_point.dtype
    y_zero_point = tf.cast(y_zero_point, tf.float32)
    y = tf.add(y, y_zero_point)
  else:  # y_zero_point default dtype = uint8
    y_dtype = tf.uint8

  y = tf.saturate_cast(y, y_dtype)

  return [y]
```

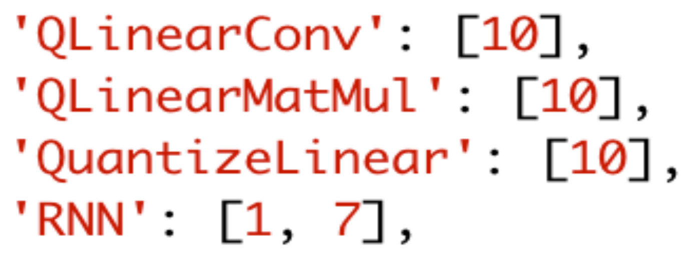
Get input name from node.inputs and input tensor from kwargs[tensor_dict]

Need to cast ‘y_zero_point’ because tf.add required both inputs in the same dtype

https://github.com/onnx/onnx-tensorflow/blob/master/onnx_tf/handlers/backend/quantize_linear.py

Register This New Handler

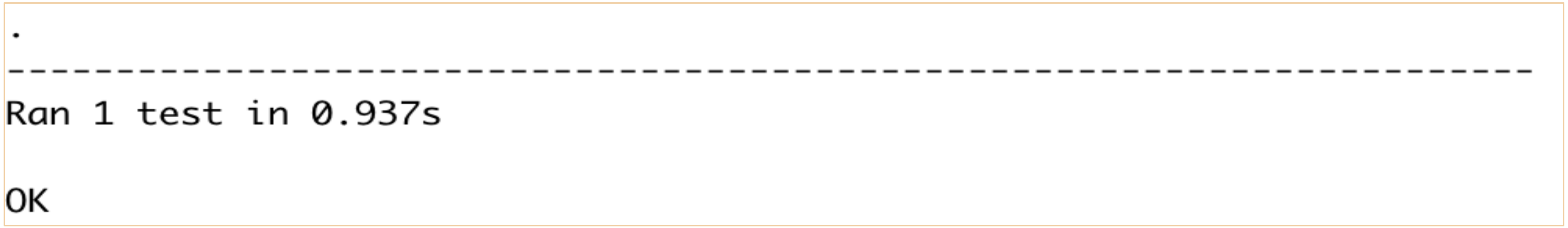
- Run onnx_tf/gen_opset.py
    - *python gen_opset.py*
- Verify version '10' is shown in the entry of QuantizeLinear in file onnx_tf/opset_version.py



```
'QLinearConv': [10],
'QLinearMatMul': [10],
'QuantizeLinear': [10],
'RNN': [1, 7],
```

Create QuantizeLinear Handler Process Walk-Through (continue)

5. Develop the handler and test it by running the test case in step 3
   - Test / debug the handler by running the unit test of quantize linear in test/backend/test_node.py
     - *python test_node.py TestNode.test_quantize_linear*



```
.
----------------------------------------------------------------------
Ran 1 test in 0.937s

OK
```

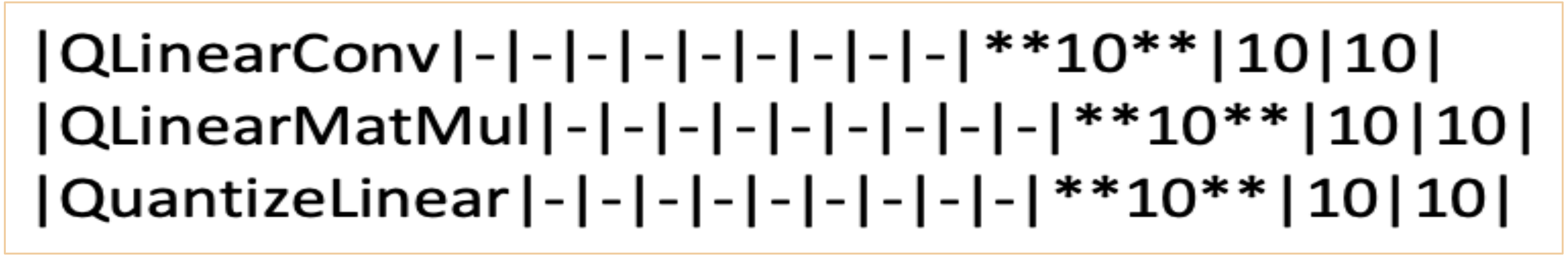
Test/Debug the Handler

- After successfully run the unit test then verify the new handler doesn’t create any new error in other tests under test/backend folder.
  - *python test_node.py*
  - *python test_dynamic_shape.py*
  - *python test_onnx_backend.py*
  - *python test_model.py*
  - *python ../test_cli.py*

  PS test_onnx_backend.py typically takes between 20 to 40 minutes to complete, depending on hardware configurations.

Update ONNX-TF Support Status Report

- Run onnx_tf/gen_status.py to update the support status
    - *python gen_status.py –v master*
- Verify QuantizeLinear entry in doc/support_status.md has been updated



```
|QLinearConv|-|-|-|-|-|-|-|-|-|**10**|10|10|
|QLinearMatMul|-|-|-|-|-|-|-|-|-|**10**|10|10|
|QuantizeLinear|-|-|-|-|-|-|-|-|-|**10**|10|10|
```

Verify All Changed Files Follow the Recommended Code Format

- Install yapf: *pip install yapf*
- Format code: *yapf -rip --style="{based_on_style: google, indent_width: 2}" $FilePath$*
- Install pylint:
    - *pip install pylint*
    - *wget -O /tmp/pylintrc https://raw.githubusercontent.com/tensorflow/tensorflow/master/tensorflow/tools/ci_build/pylintrc*
- Check format: *pylint --rcfile=/tmp/pylintrc myfile.py*

Create a Pull Request in ONNX-TF Repository

- Commit all the changes to your branch in your fork
    - *git status*
    - *git add **
    - *git commit*
    - *git push origin <your-branch-name>*
- Create PR in ONNX-TF
    - Navigate your browser to https://github.com/onnx/onnx-tensorflow
    - Click on the “Compare and pull request” button
    - Click on the ”Create Pull Request” button
    - Type a title and description of your pull request
    - Click on the “Create Pull Request“ button to submit it

Wait for Review and Address Comment

- If changes is required in your PR, modify code in your branch then run the following git commands:
  - *git status*
  - *git add **
  - *git commit --amend*
  - *git push -f origin <your-branch-name>*

Wait for Review and Address Comment (continue)

- If rebase is required, then run the following git commands
  - *git stash* # if there is uncommitted changes on the current branch then stash it else skip to next step
  - *git fetch upstream*
  - *git checkout master*
  - *git merge upstream/master*
  - *git push origin master*
  - *git checkout <your-branch-name>*
  - *git rebase origin/master*
  - *git stash pop* # if there is a stash then pop it now and run the next step
  - *git commit --amend* # if there is no change to the commit then skip to next step
  - *git show -1*
  - *git push -f origin <your-branch-name>*