ACA – Project P17

Authors: Guido Ballabio 899545 – Matteo Bellusci 898380

Professor: Cristina Silvano – **Project instructor**: Ahmet Erdem

Politecnico di Milano AY 2017-2018

Name: Tensorflow Neural Network Quantization Code: P17

Type: Programming(Python) Max Points: 12 (6+6)

Description:

For real world application, convolutional neural network(CNN) model can take more than 100MB of space and can be computationally too expensive. Therefore, there are multiple methods to reduce this complexity in the state of art. The goal of this project is to apply some neural network quantization techniques with high-level frameworks like Tensorflow and observe the effects of quantization both on the accuracy of the network and the execution performance of the neural network during inference phase. In the project we are not interested in the training phase performance. The project requires that two or more models trained for Cifar10 or MNIST dataset with Tensorflow and with possibly different quantization methodologies.

The comparison of the models will be based on the execution time, model size and cache utilization of the inference run of the neural networks that are trained. The effectiveness of comparison between different networks are essential for this project therefore it is strongly suggested for students to train networks with diverse characteristics. The inference run might be tested on CPU platforms and the cache utilization can be gathered from Linux Perf or Cachegrind tools.

Quantization techniques

Available Quantization techniques from Tensorflow [1] :

- Generation of fully 8-bit Fixed Point Quantized models through Graph Transformation
- Quantization Training through Fake Quantization technique (always 8-bit integer)

Regarding our project, here's a high level view

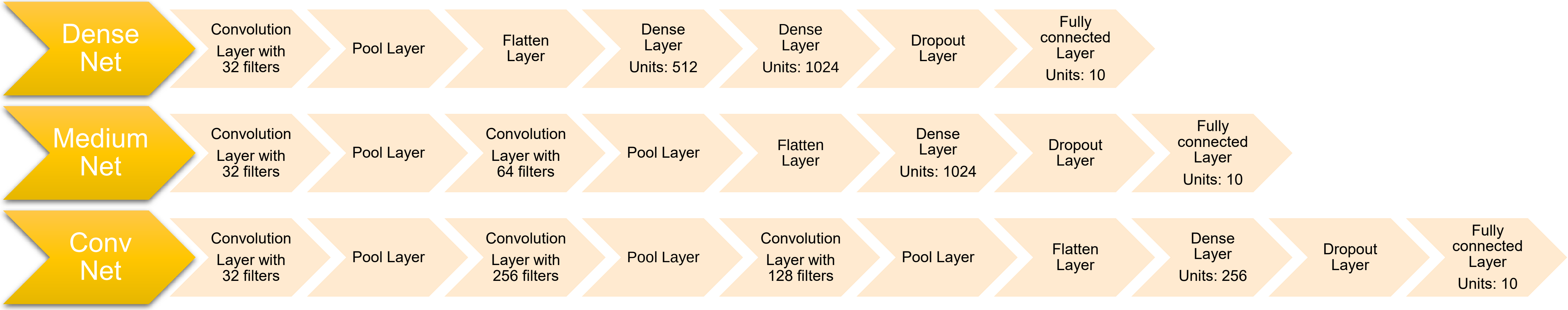
Target	
Standard nets	Quantized nets

Training	
Standard Training	Training with **Fake quantization**

Model saving	
Save Graph Model	Save after **8-bit integer Quantization** transformation

Different Cifar10 CNNs

We have defined three different cifar10 networks to carry out the comparison. Here they are:



- Dense net:
  - With the biggest fully connected layers → A lot of parameters
- Medium net:
  - The more reasonable one → average number of parameters
- Conv net:
  - Consists mostly of convolution layers (more filters) → Few parameters

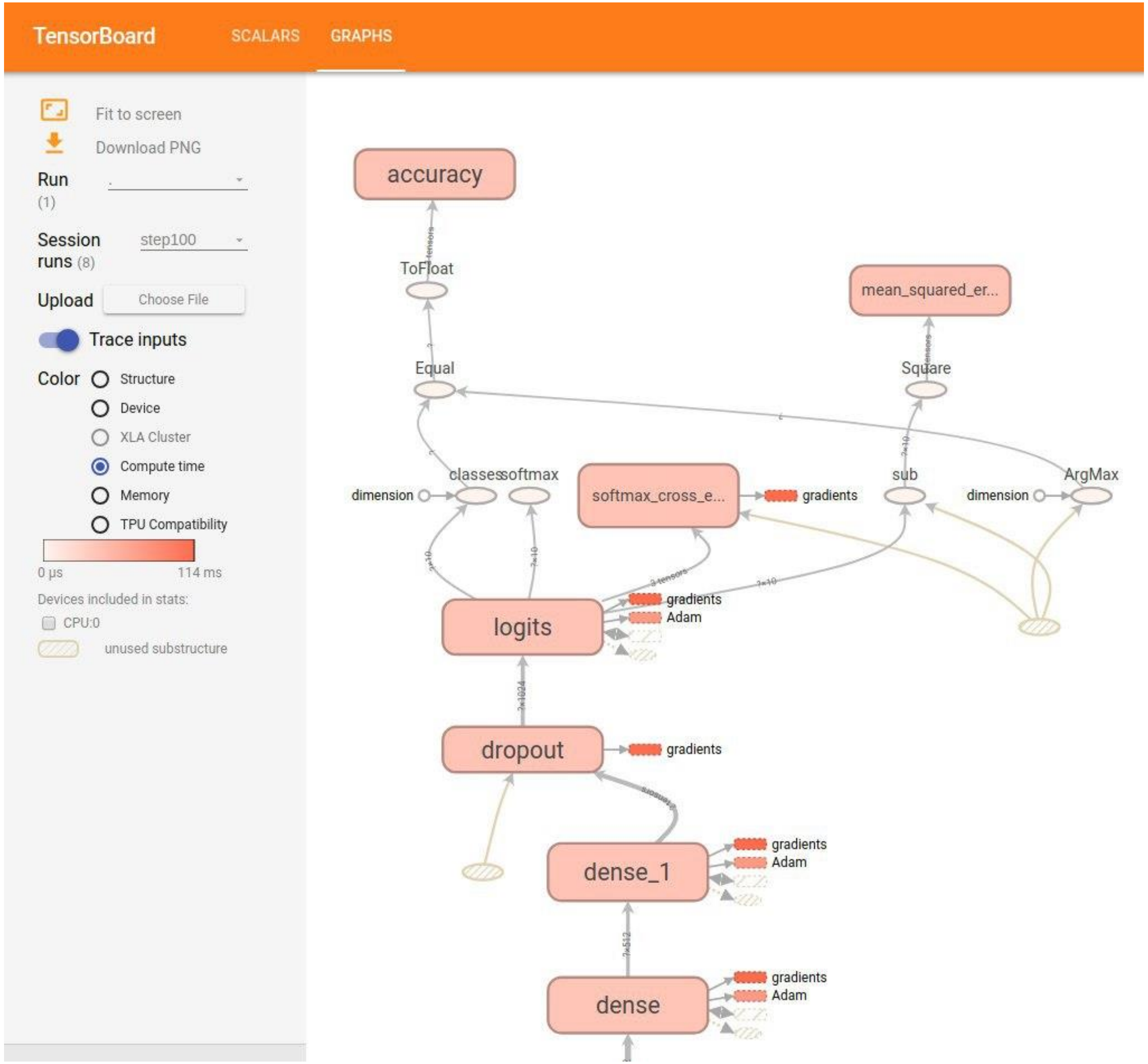
Training phase

Here there are the specifications about training phase:

- Batch size 32
- Epochs 5
  - Dictated by Early-Stopping policy.
- Dropout probability 0.5
- Validation split 0.2

Training was performed using GPU: both for standard training and quantized training **with fake quantization**.

On the right the main graph model of the medium net with information about time execution with CPU.



Model size comparison

Let’s analyze first the model size of the first net:

Dense network standard model size: 16.3 MB

Dense network quantized model size: 4.53 MB

Hence: 18.1 MB / 4.53 MB $\simeq$ 4 → **Approximately 4 Times smaller**

.

Here the comparisons of all the nets:

Network name	Standard network size	Quantized network size	SN size / QN size
Dense	18.1 MB	4.53 MB	3.996
Medium	16.3 MB	4.08 MB	3.995
Conv	4.58 MB	1.17 MB	3.915

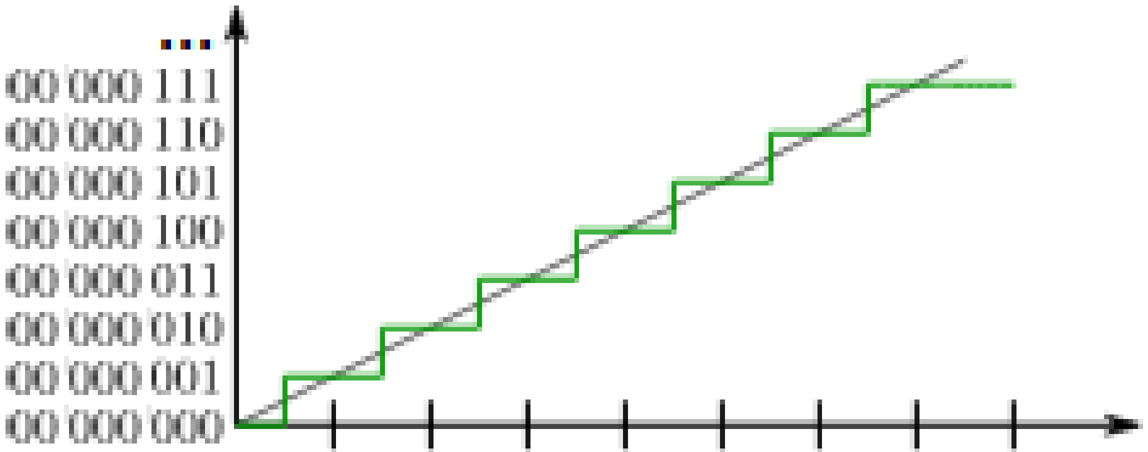
Model size comparison

Does this result make sense? **Yes!** In fact:

Standard networks use 32-bit floating-point numbers to represent their weights;

Quantized networks use a 8-bit integer representation instead.

Size is not perfectly 4 times smaller: of course a discrepancy is due to the structure of the model contained in the file, but is negligible with respect to the size of the nets' weights.



Representation	Min	Max
Float 32 bit	-3.4E+38	+3.4E+38
Integer 8 bit	-128	+127

Accuracy comparison

A smaller model size could affect the accuracy of the network. Weights can be less precise and therefore a slight worsening is expected.

Here the result of our accuracy analysis:

Network name	Standard network accuracy	Quantized network accuracy	QN accuracy / SN accuracy
Dense	0.6358	0.6220	0.98
Medium	0.6461	0.6476	$\simeq 1$
Conv	0.7345	0.7348	$\simeq 1$

→ This is a very nice result too. Accuracy is still approximately the same!

This result is reasonable because the networks in question are not extremely complicated, moreover neural networks are resilient/**robust with respect to noise**.

How we made accuracy comparison

Models are loaded from files and then are tested using the Test Dataset.

This is the code snippet in our Benchmark notebook:

Check accuracy

In [6]:
```
_, _, x_test, t_test = load_cifar10()
x_test = dataset_preprocessing_by_keras(x_test)
```

In [7]:
```
for pb_files in frozen_nets:
    for m in pb_files:
        graph = load_frozen_graph(m)
        out = predict_from_frozen(graph, [x_test], ["features"], ["classes:0", "softmax:0"])
        classes = np.concatenate([batch[0] for batch in out])
        acc = accuracy_score(np.argmax(t_test, axis=1), classes)
        print(f"{m:22}: {acc:6} accuracy")
```

```
models/dense_opt.pb   : 0.6358 accuracy
models/dense_quant.pb :  0.622 accuracy
models/conv_opt.pb    : 0.7345 accuracy
models/conv_quant.pb  : 0.7348 accuracy
models/medium_opt.pb  : 0.6461 accuracy
models/medium_quant.pb: 0.6476 accuracy
```

Performance comparison

A) Execution time performance

This step of was performed using Pyperf and the Tensorflow profiler [2].

B) Cache utilization analysis

This step of was performed using Linux Perf (as it was recommended).

There is a correlation between the two analyzes and therefore will be addressed together. The analyzes will also take into account the batch size used in the benchmark phase.

Firstly we can start seeing the execution time of the medium net (both standard and quantized). Through the Tensorflow profiler is possible to view a detailed report about the network's execution performance.

Note: Currently TF(L) quantization works only on CPUs. Hence, for comparison purpose, we have run both on CPU without Tensorflow optimization AVX2, FMA as it would have been unfair to Quantized nets!

Execution time for medium net with unit batch size

This report show the performance of the first step, with negligible effects of cache memory.

Clean cache memory

Run benchmark for a specific net

```
(myvenv3) m@m-Lenovo-ideapad-300-15ISK:~/Scrivania/aca-tensorflow$ sync; sudo sh -c "echo 3 > /proc/sys/vm/drop_caches"; python run_bench.py medium_quant
```

Standard Medium network

Memory allocations requested by the operation

Approximately 220 ms

Profile:

node name	requested bytes	total execution time	accelerator execution time	cpu execution time
Conv2D	196.61KB (100.00%, 1.14%),	106.19ms (100.00%, 47.44%),	0us (0.00%, 0.00%),	106.19ms (100.00%, 47.44%)
BiasAdd	0B (0.00%, 0.00%),	47.41ms (52.56%, 21.18%),	0us (0.00%, 0.00%),	47.41ms (52.56%, 21.18%)
Softmax	0B (0.00%, 0.00%),	31.61ms (31.38%, 14.12%),	0us (0.00%, 0.00%),	31.61ms (31.38%, 14.12%)
MatMul	4.14KB (98.86%, 0.02%),	25.81ms (17.26%, 11.53%),	0us (0.00%, 0.00%),	25.81ms (17.26%, 11.53%)
Pack	8B (98.84%, 0.00%),	7.00ms (5.73%, 3.13%),	0us (0.00%, 0.00%),	7.00ms (5.73%, 3.13%)
StridedSlice	4B (98.84%, 0.00%),	4.31ms (2.60%, 1.92%),	0us (0.00%, 0.00%),	4.31ms (2.60%, 1.92%)
MaxPool	49.15KB (98.84%, 0.28%),	1.29ms (0.67%, 0.58%),	0us (0.00%, 0.00%),	1.29ms (0.67%, 0.58%)
Relu	0B (0.00%, 0.00%),	93us (0.09%, 0.04%),	0us (0.00%, 0.00%),	93us (0.09%, 0.04%)
Const	17.04MB (98.55%, 98.55%),	67us (0.05%, 0.03%),	0us (0.00%, 0.00%),	67us (0.05%, 0.03%)
Shape	16B (0.00%, 0.00%),	49us (0.02%, 0.02%),	0us (0.00%, 0.00%),	49us (0.02%, 0.02%)

Quantized Medium network

Approximately 90 ms

Profile:

node name	requested bytes	total execution time	accelerator execution time	cpu execution time
Softmax	0B (0.00%, 0.00%),	29.52ms (100.00%, 32.85%),	0us (0.00%, 0.00%),	29.52ms (100.00%, 32.85%)
QuantizedMaxPool	12.30KB (100.00%, 0.25%),	25.26ms (67.15%, 28.10%),	0us (0.00%, 0.00%),	25.26ms (67.15%, 28.10%)
Shape	16B (99.75%, 0.00%),	15.93ms (39.05%, 17.72%),	0us (0.00%, 0.00%),	15.93ms (39.05%, 17.72%)
QuantizedMatMul	4.15KB (99.75%, 0.09%),	10.36ms (21.33%, 11.53%),	0us (0.00%, 0.00%),	10.36ms (21.33%, 11.53%)
Pack	8B (99.66%, 0.00%),	5.08ms (9.80%, 5.66%),	0us (0.00%, 0.00%),	5.08ms (9.80%, 5.66%)
QuantizedConv2D	196.62KB (99.66%, 4.06%),	1.82ms (4.15%, 2.02%),	0us (0.00%, 0.00%),	1.82ms (4.15%, 2.02%)
QuantizedBiasAdd	200.78KB (95.60%, 4.15%),	719us (2.12%, 0.80%),	0us (0.00%, 0.00%),	719us (2.12%, 0.80%)
StridedSlice	4B (91.45%, 0.00%),	545us (1.32%, 0.61%),	0us (0.00%, 0.00%),	545us (1.32%, 0.61%)
Requantize	100.44KB (91.45%, 2.07%),	285us (0.72%, 0.32%),	0us (0.00%, 0.00%),	285us (0.72%, 0.32%)
RequantizationRange	32B (89.38%, 0.00%),	115us (0.40%, 0.13%),	0us (0.00%, 0.00%),	115us (0.40%, 0.13%)
QuantizedRelu	50.20KB (89.38%, 1.04%),	55us (0.27%, 0.06%),	0us (0.00%, 0.00%),	55us (0.27%, 0.06%)
Const	64.94KB (88.34%, 1.34%),	40us (0.21%, 0.04%),	0us (0.00%, 0.00%),	40us (0.21%, 0.04%)
Min	4B (87.00%, 0.00%),	38us (0.17%, 0.04%),	0us (0.00%, 0.00%),	38us (0.17%, 0.04%)
QuantizeV2	3.08KB (87.00%, 0.06%),	34us (0.12%, 0.04%),	0us (0.00%, 0.00%),	34us (0.12%, 0.04%)
Dequantize	16.42KB (86.94%, 0.34%),	19us (0.09%, 0.02%),	0us (0.00%, 0.00%),	19us (0.09%, 0.02%)
Max	4B (86.60%, 0.00%),	17us (0.07%, 0.02%),	0us (0.00%, 0.00%),	17us (0.07%, 0.02%)
QuantizedReshape	8B (86.60%, 0.00%),	16us (0.05%, 0.02%),	0us (0.00%, 0.00%),	16us (0.05%, 0.02%)
_retval_softmax_0_0	0B (0.00%, 0.00%),	5us (0.03%, 0.01%),	0us (0.00%, 0.00%),	5us (0.03%, 0.01%)
Reshape	0B (0.00%, 0.00%),	3us (0.02%, 0.00%),	0us (0.00%, 0.00%),	3us (0.02%, 0.00%)

Execution time for medium net with unit batch size

This report show the performance of the N-step (N~150), with effects of cache memory.

Clean cache memory | **Run benchmark for a specific net**

```
(myvenv3) m@m-Lenovo-ideapad-300-15ISK:~/Scrivania/aca-tensorflow$ sync; sudo sh -c "echo 3 > /proc/sys/vm/drop_caches"; python run_bench.py medium_quant
```

Standard Medium network

Approximately 6 ms

```
Profile:
node name | requested bytes | total execution time | accelerator execution time | cpu execution time
MatMul                     6.11KB (100.00%, 0.02%),    2.60ms (100.00%, 41.81%),    0us (0.00%, 0.00%),    2.60ms (100.00%, 41.81%)
Conv2D                   290.68KB (99.98%, 1.14%),     2.58ms (58.19%, 41.45%),     0us (0.00%, 0.00%),    2.58ms (58.19%, 41.45%)
BiasAdd                        0B (0.00%, 0.00%),      425us (16.74%, 6.83%),       0us (0.00%, 0.00%),    425us (16.74%, 6.83%)
Softmax                        0B (0.00%, 0.00%),      310us (9.91%, 4.98%),        0us (0.00%, 0.00%),    310us (9.91%, 4.98%)
MaxPool                   72.67KB (98.84%, 0.28%),      94us (4.92%, 1.51%),        0us (0.00%, 0.00%),     94us (4.92%, 1.51%)
StridedSlice                   5B (98.55%, 0.00%),      88us (3.41%, 1.42%),        0us (0.00%, 0.00%),     88us (3.41%, 1.42%)
Pack                          11B (98.55%, 0.00%),      74us (1.99%, 1.19%),        0us (0.00%, 0.00%),     74us (1.99%, 1.19%)
Const                     25.19MB (98.55%, 98.55%),     22us (0.80%, 0.35%),        0us (0.00%, 0.00%),     22us (0.80%, 0.35%)
Relu                           0B (0.00%, 0.00%),       18us (0.45%, 0.29%),        0us (0.00%, 0.00%),     18us (0.45%, 0.29%)
_arg_features_0_0              0B (0.00%, 0.00%),        4us (0.16%, 0.06%),        0us (0.00%, 0.00%),      4us (0.16%, 0.06%)
Reshape                        0B (0.00%, 0.00%),        3us (0.10%, 0.05%),        0us (0.00%, 0.00%),      3us (0.10%, 0.05%)
Shape                         23B (0.00%, 0.00%),        3us (0.05%, 0.05%),        0us (0.00%, 0.00%),      3us (0.05%, 0.05%)
```

The slower layers are:

1. MatMul
2. Conv2D
3. BiasAdd

Quantized Medium network

Approximately 9 ms

```
Profile:
node name | requested bytes | total execution time | accelerator execution time | cpu execution time
QuantizedMatMul            4.74KB (100.00%, 0.09%),    5.97ms (100.00%, 65.75%),    0us (0.00%, 0.00%),    5.97ms (100.00%, 65.75%)
QuantizedConv2D          224.43KB (99.91%, 4.06%),     1.29ms (34.25%, 14.17%),     0us (0.00%, 0.00%),    1.29ms (34.25%, 14.17%)
QuantizedBiasAdd         229.17KB (95.85%, 4.15%),     912us (20.08%, 10.05%),      0us (0.00%, 0.00%),    912us (20.08%, 10.05%)
Requantize               114.64KB (91.71%, 2.07%),     296us (10.03%, 3.26%),       0us (0.00%, 0.00%),    296us (10.03%, 3.26%)
QuantizedMaxPool          14.04KB (89.64%, 0.25%),     177us (6.77%, 1.95%),        0us (0.00%, 0.00%),    177us (6.77%, 1.95%)
RequantizationRange           36B (89.38%, 0.00%),     109us (4.82%, 1.20%),        0us (0.00%, 0.00%),    109us (4.82%, 1.20%)
Softmax                        0B (0.00%, 0.00%),       58us (3.61%, 0.64%),        0us (0.00%, 0.00%),     58us (3.61%, 0.64%)
QuantizedRelu             57.30KB (89.38%, 1.04%),      54us (2.98%, 0.60%),        0us (0.00%, 0.00%),     54us (2.98%, 0.60%)
Pack                           9B (88.34%, 0.00%),      49us (2.38%, 0.54%),        0us (0.00%, 0.00%),     49us (2.38%, 0.54%)
Shape                         18B (88.34%, 0.00%),      45us (1.84%, 0.50%),        0us (0.00%, 0.00%),     45us (1.84%, 0.50%)
Const                     74.11KB (88.34%, 1.34%),      33us (1.34%, 0.36%),        0us (0.00%, 0.00%),     33us (1.34%, 0.36%)
QuantizeV2                 3.52KB (87.00%, 0.06%),      22us (0.98%, 0.24%),        0us (0.00%, 0.00%),     22us (0.98%, 0.24%)
Dequantize                18.75KB (86.94%, 0.34%),      17us (0.74%, 0.19%),        0us (0.00%, 0.00%),     17us (0.74%, 0.19%)
Max                            4B (86.60%, 0.00%),      11us (0.55%, 0.12%),        0us (0.00%, 0.00%),     11us (0.55%, 0.12%)
StridedSlice                   4B (86.60%, 0.00%),      10us (0.43%, 0.11%),        0us (0.00%, 0.00%),     10us (0.43%, 0.11%)
Min                            4B (86.60%, 0.00%),       5us (0.32%, 0.06%),        0us (0.00%, 0.00%),      5us (0.32%, 0.06%)
_arg_features_0_0              0B (0.00%, 0.00%),        3us (0.26%, 0.03%),        0us (0.00%, 0.00%),      3us (0.26%, 0.03%)
Reshape                        0B (0.00%, 0.00%),        3us (0.23%, 0.03%),        0us (0.00%, 0.00%),      3us (0.23%, 0.03%)
QuantizedReshape               9B (86.60%, 0.00%),       2us (0.20%, 0.02%),        0us (0.00%, 0.00%),      2us (0.20%, 0.02%)
_retval_softmax_0_0            0B (0.00%, 0.00%),        2us (0.18%, 0.02%),        0us (0.00%, 0.00%),      2us (0.18%, 0.02%)
```

The slower layers are the same.

Execution time for medium net with a batch size of 16

This report show the performance of the N-step (N~150), with effects of cache memory.

Clean cache memory — **Run benchmark for a specific net**

```
(myvenv3) m@m-Lenovo-ideapad-300-15ISK:~/Scrivania/aca-tensorflow$ sync; sudo sh -c "echo 3 > /proc/sys/vm/drop_caches"; python run_bench.py medium_quant
```

Standard Medium network

Approximately 45 ms

Profile:

node name	requested bytes	total execution time	accelerator execution time	cpu execution time
Conv2D	6.23MB (100.00%, 15.07%)	28.10ms (100.00%, 62.06%)	0us (0.00%, 0.00%)	28.10ms (100.00%, 62.06%)
MatMul	131.15KB (84.93%, 0.32%)	15.10ms (37.94%, 33.35%)	0us (0.00%, 0.00%)	15.10ms (37.94%, 33.35%)
MaxPool	1.56MB (84.62%, 3.77%)	862us (4.60%, 1.90%)	0us (0.00%, 0.00%)	862us (4.60%, 1.90%)
BiasAdd	0B (0.00%, 0.00%)	777us (2.69%, 1.72%)	0us (0.00%, 0.00%)	777us (2.69%, 1.72%)
Relu	0B (0.00%, 0.00%)	348us (0.98%, 0.77%)	0us (0.00%, 0.00%)	348us (0.98%, 0.77%)
Const	33.45MB (80.85%, 80.85%)	31us (0.21%, 0.07%)	0us (0.00%, 0.00%)	31us (0.21%, 0.07%)
Softmax	0B (0.00%, 0.00%)	24us (0.14%, 0.05%)	0us (0.00%, 0.00%)	24us (0.14%, 0.05%)
StridedSlice	7B (0.00%, 0.00%)	18us (0.09%, 0.04%)	0us (0.00%, 0.00%)	18us (0.09%, 0.04%)
Pack	15B (0.00%, 0.00%)	12us (0.05%, 0.03%)	0us (0.00%, 0.00%)	12us (0.05%, 0.03%)
Shape	31B (0.00%, 0.00%)	9us (0.02%, 0.02%)	0us (0.00%, 0.00%)	9us (0.02%, 0.02%)

The slower layers are:
1. Conv2D
2. MatMul
3. MaxPool

Quantized Medium network

Approximately 77 ms

Profile:

node name	requested bytes	total execution time	accelerator execution time	cpu execution time
QuantizedConv2D	6.24MB (100.00%, 23.19%)	33.67ms (100.00%, 43.03%)	0us (0.00%, 0.00%)	33.67ms (100.00%, 43.03%)
QuantizedBiasAdd	6.37MB (76.81%, 23.68%)	19.97ms (56.97%, 25.52%)	0us (0.00%, 0.00%)	19.97ms (56.97%, 25.52%)
QuantizedMatMul	131.33KB (53.14%, 0.49%)	14.25ms (31.45%, 18.21%)	0us (0.00%, 0.00%)	14.25ms (31.45%, 18.21%)
Requantize	3.19MB (52.65%, 11.84%)	4.40ms (13.25%, 5.63%)	0us (0.00%, 0.00%)	4.40ms (13.25%, 5.63%)
RequantizationRange	60B (40.81%, 0.00%)	2.67ms (7.62%, 3.42%)	0us (0.00%, 0.00%)	2.67ms (7.62%, 3.42%)
QuantizedMaxPool	390.13KB (40.81%, 1.45%)	1.33ms (4.20%, 1.71%)	0us (0.00%, 0.00%)	1.33ms (4.20%, 1.71%)
QuantizedRelu	1.59MB (39.36%, 5.92%)	923us (2.49%, 1.18%)	0us (0.00%, 0.00%)	923us (2.49%, 1.18%)
Dequantize	521.39KB (33.44%, 1.94%)	387us (1.31%, 0.49%)	0us (0.00%, 0.00%)	387us (1.31%, 0.49%)
QuantizeV2	97.54KB (31.51%, 0.36%)	384us (0.82%, 0.49%)	0us (0.00%, 0.00%)	384us (0.82%, 0.49%)
Const	127.80KB (31.14%, 0.47%)	60us (0.33%, 0.08%)	0us (0.00%, 0.00%)	60us (0.33%, 0.08%)
Max	7B (30.67%, 0.00%)	53us (0.25%, 0.07%)	0us (0.00%, 0.00%)	53us (0.25%, 0.07%)
Min	7B (30.67%, 0.00%)	32us (0.19%, 0.04%)	0us (0.00%, 0.00%)	32us (0.19%, 0.04%)
Softmax	0B (0.00%, 0.00%)	22us (0.14%, 0.03%)	0us (0.00%, 0.00%)	22us (0.14%, 0.03%)
StridedSlice	7B (30.67%, 0.00%)	20us (0.12%, 0.03%)	0us (0.00%, 0.00%)	20us (0.12%, 0.03%)
Pack	15B (30.67%, 0.00%)	13us (0.09%, 0.02%)	0us (0.00%, 0.00%)	13us (0.09%, 0.02%)
QuantizedReshape	15B (30.67%, 0.00%)	10us (0.07%, 0.01%)	0us (0.00%, 0.00%)	10us (0.07%, 0.01%)
_arg_features_0_0	0B (0.00%, 0.00%)	7us (0.06%, 0.01%)	0us (0.00%, 0.00%)	7us (0.06%, 0.01%)
Shape	31B (30.67%, 0.00%)	7us (0.05%, 0.01%)	0us (0.00%, 0.00%)	7us (0.05%, 0.01%)
Reshape	0B (0.00%, 0.00%)	6us (0.04%, 0.01%)	0us (0.00%, 0.00%)	6us (0.04%, 0.01%)

The slower layers are:
1. MatMul
2. Conv2D
3. BiasAdd

Execution time performance comparison

Here is the benchmark result from Pyperf:

Benchmark	dense_opt	dense_quant
1-batch	2.78 sec	7.92 sec: 2.85x slower (+185%)
16-batch	875 ms	1.78 sec: 2.03x slower (+103%)
64-batch	675 ms	1.40 sec: 2.08x slower (+108%)

Benchmark	conv_opt	conv_quant
1-batch	1.44 sec	4.74 sec: 3.30x slower (+230%)
16-batch	742 ms	2.78 sec: 3.75x slower (+275%)
64-batch	738 ms	2.58 sec: 3.50x slower (+250%)

Benchmark	medium_opt	medium_quant
1-batch	2.95 sec	8.53 sec: 2.89x slower (+189%)
16-batch	1.40 sec	2.76 sec: 1.97x slower (+97%)
64-batch	1.25 sec	2.38 sec: 1.91x slower (+91%)

→ **As we can see QNs seem slower.**

In the next slides we will see that this is not due to the cache memory utilization.

A lot of documentation suggests this result, in all probability, is due to **operation optimization**.

Operations computed by these layers are optimized for 32-bit floating-point numbers operators.

TF(L) Quantization it is very performing, for example, for mobile applications but currently does not guarantee the same performance on CPU like ours Intel Core i7.

See also: [3]

Cache utilization performance comparison

Here an example: the analysis of cache utilization of the medium net with batch size 16

Clear cache memory

Specific perf command for cache utilization monitoring

```
(myvenv3) m@m-Lenovo-ideapad-300-15ISK:~/Scrivania/aca-tensorflow$ sync; sudo sh -c "echo 3 > /proc/sys/vm/drop_caches"; perf stat -B -e cache-references,cache-misses,cycles,instructi
ons,branches,faults,migrations,L1-dcache-loads,L1-dcache-load-misses,L1-dcache-stores,L1-dcache-store-misses,L1-dcache-prefetches,L1-dcache-prefetch-misses,LLC-loads,LLC-load-misses,L
LC-stores,LLC-store-misses python run_bench.py dense_opt
```

Run benchmark for a specific net

SN

```
Performance counter stats for 'python run_bench.py dense_opt':

     5.141.814.877      cache-references                                      (24,96%)
     1.736.292.260      cache-misses              #   33,768 % of all cache refs      (33,29%)
   139.263.307.563      cycles                                                (33,34%)
   178.139.630.719      instructions              #    1,28  insn per cycle   (41,48%)
    22.169.771.611      branches                                              (41,56%)
         2.880.326      faults
             5.090      migrations
    50.461.019.400      L1-dcache-loads                                       (41,40%)
     3.335.599.245      L1-dcache-load-misses     #    6,61% of all L1-dcache hits    (41,46%)
    15.746.595.788      L1-dcache-stores                                      (41,49%)
   <not supported>      L1-dcache-store-misses
   <not supported>      L1-dcache-prefetches
   <not supported>      L1-dcache-prefetch-misses
       930.790.229      LLC-loads                                             (33,50%)
       140.773.161      LLC-load-misses           #   15,12% of all LL-cache hits     (33,64%)
       672.028.543      LLC-stores                                            (16,61%)
       373.928.749      LLC-store-misses                                      (16,64%)
```

QN

```
Performance counter stats for 'python run_bench.py dense_quant':

     5.906.405.210      cache-references                                      (25,13%)
     1.591.422.484      cache-misses              #   26,944 % of all cache refs      (33,39%)
   203.039.324.153      cycles                                                (33,32%)
   271.362.529.595      instructions              #    1,34  insn per cycle   (41,70%)
    30.505.935.720      branches                                              (41,73%)
         3.306.103      faults
             8.846      migrations
    52.909.028.293      L1-dcache-loads                                       (41,64%)
     2.788.499.471      L1-dcache-load-misses     #    5,27% of all L1-dcache hits    (41,67%)
    21.418.126.962      L1-dcache-stores                                      (41,63%)
   <not supported>      L1-dcache-store-misses
   <not supported>      L1-dcache-prefetches
   <not supported>      L1-dcache-prefetch-misses
       355.359.455      LLC-loads                                             (33,17%)
        49.743.632      LLC-load-misses           #   14,00% of all LL-cache hits     (33,24%)
       517.825.747      LLC-stores                                            (16,73%)
       347.624.418      LLC-store-misses                                      (16,74%)
```

$HitRate_{QN} / HitRate_{SN} \simeq (1 - 27\%)/(1 - 34\%) \simeq 1.11$

Hit rate for QNs network seems higher. Let’s see the complete comparison…

Cache utilization performance comparison

Here is the complete cache utilization result:

Batch size	Dense SN	Dense QN	Med. SN	Med. QN	Conv SN	Conv QN
1	57,10,43	14,9,5	45,10,19	13,9,4	34,9,30	14,6,12
16	34,7,15	27,5,14	40,7,27	25,5,13	33,7,27	31,3,28
64	42,6,35	39,4,24	44,6,41	36,5,22	40,7,34	45,3,30

- The first number is the Total Cache miss rate (in %, e.g. 57%)
- The second number is the L1 Cache miss rate (in %, e.g. 10%)
- The third number is the LLC Cache miss rate (in %, e.g. 43%)

It's clear from these results that **QNs have a higher hit rate**!

Since fetching 8-bit values only requires 25% of the memory bandwidth of floats, it's easier for cache to avoid bottlenecks for RAM access.

IPC performance comparison

Here is an additional comparison gathered from Linux Perf. This table show how many instruction per clock cycle are computed.

Batch size	Dense SN	Dense QN	Med. SN	Med. QN	Conv SN	Conv QN
1	1.22	1.58	1.31	1.54	1.38	1.43
16	1.28	1.34	1.28	1.37	1.36	1.25
64	1.39	1.26	1.35	1.33	1.25	1.18

In red we identify when Quantization is worse.

The higher batch size, the lower is the gain in using quantization.

Also the gain seems to be inversely correlated with the net size probably due to how parallelizable convolutions are.

References

1) Tensorflow quantization documentation: https://www.tensorflow.org/performance/quantization
2) Tensorflow profiler documentation: https://www.tensorflow.org/api_docs/python/tf/profiler/Profiler
3) Discussion about Quantization time performance: https://github.com/tensorflow/tensorflow/issues/2807

Our GitHub repository:

https://github.com/GuidoBallabio/aca-tensorflow