Slow quantized graph #2807

Closed **changyun79** opened this issue on 11 Jun 2016 · 39 comments

Assignees 

changyun79 commented on 11 Jun 2016

1. On Ubuntu 15.10 with CUDA 7.5, cuDNN 7.0, tensorflow-0.9.0rc0, ran "tensorflow/examples/label_image/" application by taking inception-v3 graph and roughly measure the elapsed time.

2. Then take "tensorflow/contrib/quantization/tools:quantize_graph" to quant inception-v3, rebuilt application by giving

```
"//tensorflow/contrib/quantization:cc_ops",
"//tensorflow/contrib/quantization/kernels:quantized_ops",
```

into "tensorflow/examples/label_image/BUILD" and redo the same classification and measure the time.

Before/After quantization, elapsed time were 6 seconds vs. 17 seconds, i.e. quantization doubled the inference time?

The results looks ok as below so I think I was running it correctly.
Before

- military uniform (866): 0.647299
- suit (794): 0.0477195
- academic gown (896): 0.0232407

- suit (794): 0.0248454
- bow tie (817): 0.0171362
- bolo tie (940): 0.0171362
- academic gown (896): 0.0164432

My tensor flow was built as CPU only. Have also tried to enable GPU while the timing didn't change. Do we know what the expected performance would be?

13

vrv assigned **petewarden** on 11 Jun 2016

vrv commented on 11 Jun 2016 — Contributor

Maybe it's possible that the reference (slow) implementations are used when not running on Android/ARM. **@petewarden** would know more.

changyun79 commented on 11 Jun 2016 — Author

Thanks for the response, vrv.

I am running on Intel i7-3770 with Nvidia GM200 [GeForce GTX TITAN X] GPU (I am profiling CPU only anyways). The profiling is very rough as I am just taking bash timer. The complete script, which actually comes from Pete's blog is as following.

```
bazel build tensorflow/contrib/quantization/tools:quantize_graph
bazel build tensorflow/examples/label_image:label_image

curl http://download.tensorflow.org/models/image/imagenet/inception-2015-12-05.tgz -o
/tmp/inceptionv3.tgz
tar xzf /tmp/inceptionv3.tgz -C ./tensorflow/examples/label_image/data/

echo "Quantizing Incepton-v3 graph as 8bits mode"
bazel-bin/tensorflow/contrib/quantization/tools/quantize_graph \
--input=./tensorflow/examples/label_image/data/classify_image_graph_def.pb \
```



```
--graph=./tensorflow/examples/label_image/data/classify_image_graph_def.pb
```

```
ENDTIME=$(date +%s)
echo "It takes $(($ENDTIME - $STARTTIME)) seconds to complete this task...\n\n"


echo "Quantized as 8 bits"
STARTTIME=$(date +%s)
#command block that takes time to complete...
#........
./bazel-bin/tensorflow/examples/label_image/label_image \
--graph=./tensorflow/examples/label_image/data/quantized_graph_eightbit.pb \
--input_width=299 \
--input_height=299 \
--input_mean=128 \
--input_std=128 \
--input_layer="Mul:0" \
--output_layer="softmax:0"
ENDTIME=$(date +%s)
echo "It takes $(($ENDTIME - $STARTTIME)) seconds to complete this task...\n\n"
```

changyun79 commented on 11 Jun 2016 — Author

By doing this,

```
model_filename = os.path.join(
        "./tensorflow/tensorflow/examples/label_image/data", 'quantized_graph_eightbit.pb')
    print model_filename
    with gfile.FastGFile(model_filename, 'rb') as f:
      graph_def = tf.GraphDef()
      graph_def.ParseFromString(f.read())
      writer=tf.train.write_graph(graph_def, './tensorflow/tensorflow/examples/label_image/data',
'quantized_graph.pb')
```

I print out the graph and load into tensorboard to visually check it and looks 2 pool layers

- pool between conv_2 and conve_3
- pool_1 between conv_4 and mixed

  goes away in the "quantized" version graph.

Is there any reason? I am quite new to tensorflow so there could be something wrong on my side

changyun79 commented on 11 Jun 2016 Author

I just tried giving -pg as

```
bazel build --copt="-pg" --cxxopt="-pg" tensorflow/examples/label_image:label_image
```

and rerun the quantized graph after rebuild. It then took 245 seconds to finish the classification. I did not find gprof log file produced and results even did not match which confused me:

- military uniform (866): 0.579775
- suit (794): 0.0243188
- bolo tie (940): 0.0200896
- academic gown (896): 0.0179139
- Windsor tie (935): 0.010099

Any recommended way to do proper profiling?

Thanks.

futurely commented on 17 Jun 2016

What are the results running on GPU?

changyun79 commented on 17 Jun 2016 Author

After quant, on GPU:
military uniform (866): 0.703474
suit (794): 0.0248454
bow tie (817): 0.0171362
bolo tie (940): 0.0171362
academic gown (896): 0.0164432

petewarden commented on 18 Jun 2016 Member

Can you try using https://github.com/tensorflow/tensorflow/tree/master/tensorflow/tools/benchmark and let me know what you get? There are some known optimizations for eight-bit CPU we're working on, but it would be useful to see where the time is going from the profiling logs you'll get from that tool.

changyun79 commented on 18 Jun 2016 Author

sure. The machine has launched some over weekend testing so I will do it on coming Monday. Thanks.

changyun79 commented on 20 Jun 2016 Author

`

quant_inceptionv3.txt

`

Attached output from benchmarking on quantized inception v3.

aselle added **triaged** **stat:awaiting response** and removed **stat:awaiting response** labels on 27 Jun 2016

zt706 commented on 12 Jul 2016

I have the same problem and my operation is similary to "changyun79". I test inception_v3 on my computer and the results were:
Before/After quantization, elapsed time were 36 seconds vs. 108 seconds per 10 images
Before/After quantization, my trained inception_v3 size were 92M vs. 22M

aselle removed the **triaged** label on 28 Jul 2016

MingSun-Tse commented on 5 Aug 2016 • edited

my result is (for one image)

- inception-v3 : 8s
- inception-v3-quantized : 20s
- inception-v3-stripped : 7s

with

- tf0.9.0-gpu
- TeslaK40-cuda7.5-cudnn5.0
- ubuntu1404 LTS

huyong1109 commented on 5 Aug 2016

I tried a small network on GPU and get similar results. (https://stackoverflow.com/questions/38775675/why-my-tensorflow-network-becomes-slower-after-applying-the-quantization-tools-o
) It may because GeForce GTX has less integer computing cores?

vzvzx commented on 1 Sep 2016

i get the same result, any one had solved?

poxvoculi mentioned this issue on 1 Sep 2016

rieram commented on 8 Sep 2016

I do get the same slow down on a single CPU. I ran Intel vtune profiler (for CPUs) on the same test image and it seems that its just not optimized. There needs to be better optimization for gemmlowp. Comparing it with the FP32 implementation that is optimized to a certain extent does better.

andrewharp mentioned this issue on 19 Sep 2016

Android version of TensorFlow is missing dequantize operation #4346

Closed

isabel-schwende mentioned this issue on 28 Sep 2016

Why is quantized graph inference takes much more time than using the original graph? #4434

Closed

sallamander commented on 12 Oct 2016

Has anybody verified that quantized computations are taking place on their GPU? I have not been able to get them to run on the GPU. Everything runs okay - GPU memory is being reserved and blocked off, but the computation is still taking place on the CPU.

@petewarden it appears that maybe the quantized ops don't have GPU support, or that it is not optimized quite yet. Any insight on this, or anything in the works on this front?

petewarden commented on 17 Oct 2016 Member

Quantized ops currently only work on the CPU, because most GPUs don't support eight-bit matrix multiplications natively. I have just seen that the latest TitanX Pascal cards offer eight-bit support though, so I'm hoping we will be able to use that in the future.

@petewarden Should we expect the quantized model to speed up inference as of `0.11.0rc1` on mobile? I just tried one locally and it seems slower as also mentioned in #4434. Thanks

eaigner commented on 2 Dec 2016 • edited Contributor

@petewarden If I quantize the graph and run it on iOS (CPU), I too get about 3 times worse performance than running the unquantized version.

lihungchieh commented on 12 Jan 2017

I met the same problem, the quantized model is much slower than the unquantized one, are there any solution now?

drpngx commented on 24 Jan 2017 Contributor

Anyone tried on TitanX?

@lihungchieh did you try compiling the latest code with `--config=opt` ? It should have `-march=native` , perhaps that helps on CPU.

aselle added the **stat:awaiting response** label on 27 Jan 2017

bhack mentioned this issue on 10 Feb 2017

Quantization code performance. tiny-dnn/tiny-dnn#552

@petewarden I ran the benchmark mentioned above on 32-bit ARM using the google inception model.But I got the result below, it shows that the 8-bit quantized graph is much slower than the 32-bit float one. Is this result the reasonable one and what's the performance of the latest version on 32-bit ARM. This really bothered me a lot. Looking forward to your response. Thanks.

fp result:

native : stat_summarizer.cc:392 ============================== Top by Computation Time ==============================
native : stat_summarizer.cc:392 [node type] [start] [first] [avg ms] [%] [cdf%] [mem KB] [Name]
native : stat_summarizer.cc:392 Conv2D 124.403 274.567 165.023 14.748% 14.748% 2408.448 conv2d2_pre_relu/conv
native : stat_summarizer.cc:392 Conv2D 472.443 103.640 84.950 7.592% 22.340% 602.112 mixed3b_3x3_pre_relu/conv
native : stat_summarizer.cc:392 Conv2D 0.298 133.612 74.971 6.700% 29.041% 3211.264 conv2d0_pre_relu/conv
native : stat_summarizer.cc:392 Conv2D 913.577 55.950 47.739 4.267% 33.307% 250.880 mixed4e_3x3_pre_relu/conv
native : stat_summarizer.cc:392 Conv2D 366.220 55.346 45.836 4.096% 37.403% 401.408 mixed3a_3x3_pre_relu/conv
native : stat_summarizer.cc:392 Conv2D 841.525 50.361 38.759 3.464% 40.867% 225.792 mixed4d_3x3_pre_relu/conv
native : stat_summarizer.cc:392 Conv2D 769.854 41.145 33.240 2.971% 43.838% 200.704 mixed4c_3x3_pre_relu/conv
native : stat_summarizer.cc:392 Conv2D 567.045 40.916 31.938 2.854% 46.692% 301.056 mixed3b_5x5_pre_relu/conv
native : stat_summarizer.cc:392 LRN 300.748 25.291 25.403 2.270% 48.963% 2408.448 localresponsenorm1
native : stat_summarizer.cc:392 Conv2D 706.635 29.413 24.777 2.214% 51.177% 175.616 mixed4b_3x3_pre_relu/conv
native : stat_summarizer.cc:392
native : stat_summarizer.cc:392 ============================== Summary by node type ==============================
native : stat_summarizer.cc:392 [Node type] [count] [avg ms] [avg %] [cdf %] [mem KB]
native : stat_summarizer.cc:392 Conv2D 57 900.237 80.466% 80.466% 12904.640
native : stat_summarizer.cc:392 MaxPool 13 76.670 6.853% 87.319% 5666.752
native : stat_summarizer.cc:392 BiasAdd 58 56.362 5.038% 92.357% 12908.672
native : stat_summarizer.cc:392 LRN 2 34.084 3.047% 95.403% 3211.264
native : stat_summarizer.cc:392 Relu 57 21.780 1.947% 97.350% 12904.640
native : stat_summarizer.cc:392 MatMul 1 18.691 1.671% 99.020% 4.032
native : stat_summarizer.cc:392 Concat 9 8.364 0.748% 99.768% 4939.200
native : stat_summarizer.cc:392 Const 118 1.951 0.174% 99.942% 0.000
native : stat_summarizer.cc:392 AvgPool 1 0.387 0.035% 99.977% 4.096
native : stat_summarizer.cc:392 Softmax 1 0.128 0.011% 99.988% 4.032
native : stat_summarizer.cc:392 <> 1 0.108 0.010% 99.998% 0.000
native : stat_summarizer.cc:392 Reshape 1 0.021 0.002% 100.000% 0.000
native : stat_summarizer.cc:392
native : stat_summarizer.cc:392 Timings (microseconds): count=50 first=1460227 curr=1078211 min=1051457

native : stat_summarizer.cc:392 ============================== Top by Computation Time ==============================
native : stat_summarizer.cc:392 [node type] [start] [first] [avg ms] [%] [cdf%] [mem KB] [Name]
native : stat_summarizer.cc:392 QuantizedConv2D 236.817 246.238 240.406 14.122% 14.122% 2408.456 conv2d2_pre_relu/conv_eightbit_quantized_conv
native : stat_summarizer.cc:392 QuantizedConv2D 779.869 125.167 126.172 7.412% 21.534% 602.120 mixed3b_3x3_pre_relu/conv_eightbit_quantized_conv
native : stat_summarizer.cc:392 QuantizedConv2D 8.386 119.831 104.379 6.131% 27.665% 3211.272 conv2d0_pre_relu/conv_eightbit_quantized_conv
native : stat_summarizer.cc:392 QuantizedConv2D 1431.287 70.142 69.877 4.105% 31.770% 250.888 mixed4e_3x3_pre_relu/conv_eightbit_quantized_conv
native : stat_summarizer.cc:392 QuantizedConv2D 628.304 59.150 59.561 3.499% 35.268% 401.416 mixed3a_3x3_pre_relu/conv_eightbit_quantized_conv
native : stat_summarizer.cc:392 QuantizedConv2D 1291.839 57.775 56.634 3.327% 38.595% 225.800 mixed4d_3x3_pre_relu/conv_eightbit_quantized_conv
native : stat_summarizer.cc:392 QuantizedConv2D 1191.274 44.160 44.362 2.606% 41.201% 200.712 mixed4c_3x3_pre_relu/conv_eightbit_quantized_conv
native : stat_summarizer.cc:392 QuantizedConv2D 929.076 41.119 41.609 2.444% 43.645% 301.064 mixed3b_5x5_pre_relu/conv_eightbit_quantized_conv
native : stat_summarizer.cc:392 QuantizedConv2D 1102.009 34.599 34.248 2.012% 45.657% 175.624 mixed4b_3x3_pre_relu/conv_eightbit_quantized_conv
native : stat_summarizer.cc:392 QuantizedConv2D 1650.254 30.406 31.038 1.823% 47.480% 75.272 mixed5b_3x3_pre_relu/conv_eightbit_quantized_conv
native : stat_summarizer.cc:392 ============================== Summary by node type ==============================
native : stat_summarizer.cc:392 [Node type] [count] [avg ms] [avg %] [cdf %] [mem KB]
native : stat_summarizer.cc:392 QuantizedConv2D 57 1217.166 71.515% 71.515% 12905.096
native : stat_summarizer.cc:392 Requantize 116 191.868 11.273% 82.788% 6455.264
native : stat_summarizer.cc:392 QuantizedBiasAdd 58 95.586 5.616% 88.405% 12909.136
native : stat_summarizer.cc:392 QuantizedMaxPool 13 42.846 2.517% 90.922% 1416.792
native : stat_summarizer.cc:392 QuantizedConcat 9 34.559 2.031% 92.953% 1234.872
native : stat_summarizer.cc:392 RequantizationRange 116 33.522 1.970% 94.922% 0.928
native : stat_summarizer.cc:392 LRN 2 31.049 1.824% 96.746% 3211.264
native : stat_summarizer.cc:392 QuantizedMatMul 1 26.198 1.539% 98.286% 4.040
native : stat_summarizer.cc:392 QuantizeV2 3 9.296 0.546% 98.832% 953.368
native : stat_summarizer.cc:392 QuantizedRelu 57 9.132 0.537% 99.368% 3226.616
native : stat_summarizer.cc:392 Const 356 4.124 0.242% 99.611% 0.000
native : stat_summarizer.cc:392 Dequantize 3 2.933 0.172% 99.783% 3215.296
native : stat_summarizer.cc:392 Min 3 1.459 0.086% 99.869% 0.024
native : stat_summarizer.cc:392 Max 3 1.315 0.077% 99.946% 0.024

native : stat_summarizer.cc:392 Timings (microseconds): count=50 first=1739035 curr=1690331 min=1684873 max=1782461 avg=1.70236e+06 std=20766
native : stat_summarizer.cc:392 Memory (bytes): count=50 curr=45537792(all same)
native : stat_summarizer.cc:392 812 nodes defined 804 nodes observed

 **prb12** added **stat:awaiting tensorflower** and removed **stat:awaiting response** labels on 8 Mar 2017

 **maydaygmail** commented on 13 Mar 2017 • edited

I have the similar problem. I quantify my own cnn model, before quantization, the size of model is 11M, and inference takes 72 ms per 128 small images; after quantization, size become 2.8M, but takes 236ms. I run the experiment in OS X ,2.6 GHz Intel Core i5.

 **sundw2014** commented on 29 Mar 2017

I had the similar problem. Finally, I found that there are some comments in
`tensorflow/tensorflow/core/kernels/quantized_conv_ops.cc`

```
  // This means that multiple ops can't be run simultaneously on different
      // threads, because we have a single shared resource. The platforms this is
      // aimed at have intra-op parallelism as their focus though, so it shouldn't
      // be an issue.
      mutex_lock lock_buffer(im2col_buffer_resource->mu);
      core::ScopedUnref unref_buffer(im2col_buffer_resource);
      T1* im2col_buffer = im2col_buffer_resource->data;
```

So, to get a good performance, you should set `inter_op_parallelism_threads=1` when you use `QuantizedConv2D` op.
But, even you do that, you will find that although the conv ops get faster because of the use of `gemmlowp`, quantization will add some new ops like `RequantizationRange`, `Requantize` to your model, which are very time-consuming. So, the quantized model will still be slower than the original one.

I have same issue on Inception V3.

petewarden commented on 16 Jun 2017 Member

The quantization is aimed at mobile performance, so most of the optimizations are for ARM not x86. We're hoping to get good quantization on Intel eventually, but we don't have anyone actively working on it yet.

 1

petewarden removed the **stat:awaiting tensorflower** label on 16 Jun 2017

clumsydzd commented on 11 Jul 2017

Any progress? After quantize, network is 3 times slower with bad performance at accuracy

ivamluz commented on 14 Jul 2017

Same issue described by **@clumsydzd** here.

passerbyd commented on 8 Sep 2017

what's the expected performance of inception-v3 on arm-v7? **@petewarden**

jinxin0924 commented on 13 Oct 2017 Contributor

@drpngx,
Any example to use quantized graph with GPU for inference?

drpngx commented on 13 Oct 2017 Contributor

/CC **@aselle**

@nealwu do we have something in the models repo?

nealwu commented on 13 Oct 2017 Contributor

Not that I know of. Maybe **@suharshs** would know.

suharshs commented on 13 Oct 2017 Contributor

Not yet. I am not aware of progress on GPU support.

merryHunter commented on 31 Oct 2017

Is there any news whether GPUs support eight-bit operations? Or any other optimization suggestions from tensorflow side?

It has been 14 days with no activity and this issue has an assignee.Please update the label and/or status accordingly.

tensorflowbutler commented on 5 Jan 2018 Member

Nagging Assigneee: It has been 14 days with no activity and this issue has an assignee. Please update the label and/or status accordingly.

MarkSonn commented on 16 Jan 2018 • edited

Does anyone know which GPUs are optimised for 8 bit calculations? The best info I have found so far is this article which compares the floating point performance between 64 bit and 16 bit operations. I'm sort of extrapolating that GPUs which had a decrease in performance on 16 bits would have a further decrease with 8 bits, and those with an increase *might* have a further increase on 8 bits. I hope this helps, but I would appreciate confirmation from someone :)

petewarden commented on 29 Jan 2018 Member

We are focusing our eight-bit efforts on TF Lite (visible at tensorflow/contrib/lite), so we aren't expecting TensorFlow's quantized performance to improve in cases where it's not currently fast. These tend to be on x86 platforms (we're concentrating on ARM performance for mobile), and for models that use ops that we don't have quantized implementations for (which is most models outside a few vision-related ones we've optimized for).

Since we're not likely to see changes in this area soon, I'm closing this as infeasible. Pull requests or other help in this area would be very welcome of course!


1

CPU slowdown with Quantized Eight bit graphs #5757

Closed

Quantization make graph slower during inference. #13939

Closed

jason9075 added a commit to jason9075/tensorflow_playground that referenced this issue on 5 Mar 2019

嘗試測試h5轉lite版 ··· 0609929

avinash31d commented on 10 Apr

As far as I understand from this entire thread is there are no plans to put 8bit on x86 or desktop CPUs. I tried with transform graph as well as building and running TfLite on windows but still its very slow when it comes to quantized model. And if there no plans to support that, what should we users of tensorflow do? How do we get this thing supported?

Assignees

petewarden

Labels

None yet

Projects

None yet

Milestone

No milestone

30 participants