Korea Advanced Institute of Science and Technology

CS492 Special Topics in Computer Science
Systems for Machine Learning
Spring 2020

Final Exam

Jongse Park
(jspark@casys.kaist.ac.kr)

Due: June 18th 11:59:59 KST
(Late submission is strictly not allowed)

Quantization for Deep Learning is a promising research direction for efficient DNN inference by offering low latency and high energy efficiency *while* compromising the accuracy loss due to the lower bit width. The state-of-the-art research works in Machine Learning community have explored solutions to quantize a DNN model without hurting the inference accuracy, mostly through a re-training process, which requires a substantial amount of compute resource (e.g., hundreds of server-class GPUs). But wait, do we really need such re-training? In this exam, you will develop a *naively-quantized* Convolution, using a diverse set of precisions and various hardware platforms, and analyze the tradeoff between the performance benefits and accuracy loss.

(*Before you start*) Please very carefully read followings before you start the exam.

- Do not copy from others. You will get a F if we find out that your code or answers are copies of others, according to the University policy.
- Needless to say, this is an individual project. Do not copy your teammate's code or answers. However, it is okay to refer to the code snippets of past projects.
- You need to implement your code in the given docker environment.
- All programs must be written in C or C++. Create a *separate* directory for each problem. The directory should contain both .c(pp) file and Makefile for the given problem. The names of directories are *prob[problem number]* (i.e., prob1, prob2, prob3, and prob4).
- Compile your code with the optimization level 3. (-O3)
- Create a zip file that contains (1) a PDF that has the written answers, and (2) a directory, *code*, which has the prob1, prob2, prob3, and prob4 directories.
- Submit the zip file on KLMS

1. [10 pt] *Develop a Convolution function in C/C++*

   Recall what you did in the second project. You constructed a graph for models that constitute five different types of models and implemented their arithmetic operations (i.e., run).

   ① Write a program that performs the arithmetic operations for *Convolution* layer in C/C++. This program takes as inputs two binary files: (1) input_tensor.bin, and (2) kernel_tensor.bin. Three pairs of (input_tensor.bin, kernel_tensor.bin) will be provided, which were generated by a real inference call using the pre-trained ResNet-50 and an input image of elephant. Your program should generate an output file, output_tensor.bin. These three binaries will have the same format, of which first 16 bytes are four integer numbers that represent the four-dimensional tensor shape.

   For inputs, the 16 bytes are for (N, H, W, C), where N is the batch size, H is the height, W is the width, and C is the channel. For kernels, the 16 bytes are for (KH, KW, OC, IC), where KH is the kernel height, KW is the kernel width, OC is the output channels, and IC is the input channels. All of these values are integer. The following N*H*W*C*4 bytes will contain the single-precision floating point (FP32) numerals, which are the values in the 1-D flattened array of input tensor. Assume the padding mode is always the "SAME", which means the shape of input tensor is maintained in the output tensor. Assume the stride is always 1. Use solely the primitive scalar operations (i.e., +, -, *, /).

   Finally, the running command should be:

   ```
   $ ./convolution input_tensor.bin kernel_tensor.bin
   ```

   ② Measure the elapsed time *exclusively* for running the convolution operations. Do not include the pre- and post-processing time in your measurement (e.g., file read/write). You can use any libraries (e.g., *time.h*) for the time measurement.

2. [15 pt] *Quantize the operands in a lower precision and analyze the performance-accuracy tradeoff*

① Take your code from Problem 1 and build upon it. Write a program that runs the same set of compute operations compared to Problem 1 *while* it allows different precision modes, INT32, INT16, and INT8. To enable such features, you would need to convert the types of data (i.e., input and kernel tensors) from FP32 to the integer types. An easiest and naïve way of achieving this goal is to simply multiple an appropriate constant to all the values, perform the integer operations, and convert them back to the FP32 values.

Consider the following example. Assume you would like to perform $C_{fp32} = A_{fp32} * B_{fp32}$ where they are all FP32 values. Pick an appropriate constant, S. Multiple S to the FP32 values and typecast them to integer so that all the values are now integers: $A_{int32} = (int)(A_{fp32} * S)$ and $B_{int32} = (int)(B_{fp32} * S)$. The multiplication of these two integer values, (A_{int32} and B_{int32}), is an integer operation. After the computation is done, we can convert the output value back to the FP32 type by dividing the output with S^2: $C_{fp32} \approx (A_{int32} * B_{int32}) / S^2 = ((int)(A_{fp32} * S) * (int)(B_{fp32} * S)) / S^2$. The question is, which S value should you use to minimize the accuracy loss?

Finally, the running command should be:

./convolution input_tensor.bin kernel_tensor.bin **[32/16/8]**

② Measure the accuracy loss introduced by the quantization using a metric, normalized root-mean-square error (NRMSE). NRMSE of values between two sets *X* and *Y* is calculated as follow,

$$\text{NRMSE} = \frac{\sqrt{\sum_i (x_i - y_i)^2 / |X|}}{y_{max} - y_{min}}, x_i \in X, y_i \in Y$$

where x_i is an approximated value of y_i.

③ Report which value you used for S and justify why you picked the value using the NRMSE numbers you obtained for each S.

④ Similar to Problem 1, measure the elapsed time for running convolution operations. In addition, measure the elapsed time for quantization overhead.

3. [15 pt] *CPU vectorization with lower precision (Word limit: 800)*

① Take your code from Problem 1 and build upon it. Use *pthreads* to parallelize the code and AVX2 instructions to vectorize with three different precisions for the operands: FP32, INT32, and INT16. You can use any AVX2 instructions for this purpose.

Finally, the running command should be:

./convolution input_tensor.bin kernel_tensor.bin **[FP32/INT32/INT16]**

② Measure the accuracy loss using the same way you did in Problem 2

③ Measure the performance using the same way you did in Problem 2.

4. [10 pt] *GPU vectorization*

① Take your code from Problem 1 and build upon it. Use CUDA (i.e., no cuBLAS/cuDNN) to vectorize the convolution operations.

Finally, the running command should be:

./convolution input_tensor.bin kernel_tensor.bin

② Measure the performance using the same way you did in Problem 1.

5. [30 pt] *Performance-Accuracy Tradeoff Analysis and Discussion (Word limit: 800)*

① Collect the measurement results from Problem 1/2/3/4. Plot the results into graphs and present the performance speedup and accuracy loss.

② Analyze the tradeoff and discuss your rationales. Refer to the discussed papers. Refer to any online articles, news, etc, if necessary. The possible discussion points include but not limited to:

i. What option seems to be the best and why?

ii. What factors do we need to consider to make a call?

iii. Would the decision change if you consider their different energy efficiencies?

iv. Is the naïve quantization justified? How much accuracy loss is acceptable?