Auto-generating Quantized DNN Kernels using TVM - Mila Lukic

The goal of this project is to use TVM compile a quantized ResNet50 down to a executable function; for simplicity, it's not yet necessary to test the accuracy of the compiled model.

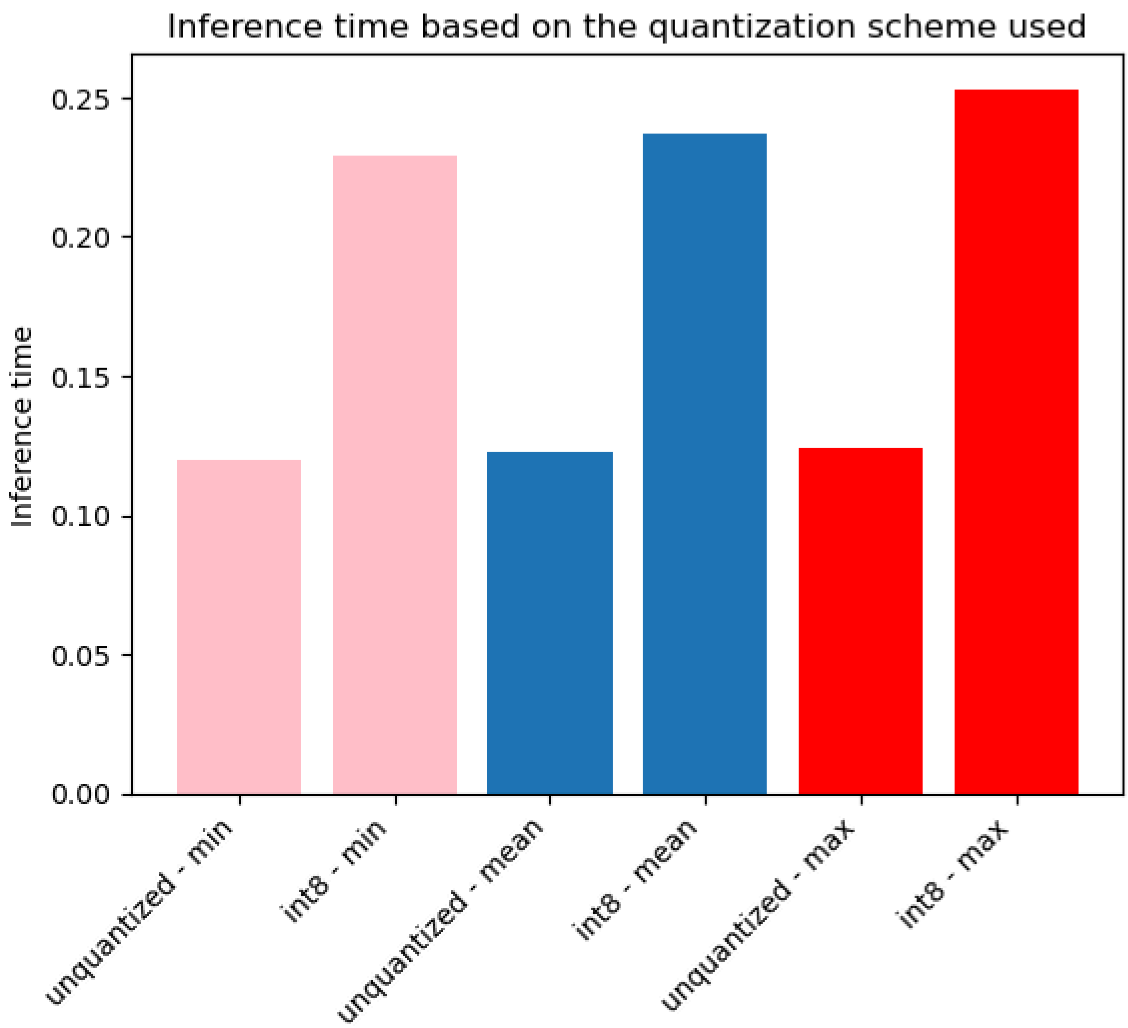
This is the github link to the implementation:
https://github.com/milalukic/tvm-quantization/blob/main/tvm_quantization_mila.ipynb

In this document, I am going to answer the “non-coding” questions posed in the assignment.

Model execution time comparison

Use TVM's utility functions to benchmark the inference time of the quantized model vs. the un-quantized model.

In this task we will not try to maximize the performance of the quantized DNN, but if there is no speedup, you should try to understand it and formulate a guess.



We can see that the unquantized DNN is faster in terms of inference time (when we look at all of the benchmarks - min, max, as well as mean). This is unexpected - the quantization to a smaller precision is supposed to make the DNN faster, but less accurate.

After some research, I found out these possible reasons for this model to be slower:

- **Overhead from Quantization**: While the lower precision reduces computation for the core operations, these calibration and conversion might outweigh the gains on some hardware.
- **Hardware Acceleration Not Optimized**: If my hardware is not specifically optimized for running low-precision models, it might not be able to fully exploit the benefits of quantization.

Considering the fact that I've got this warning message when I compiled the unquantized DNN:

```
One or more operators have not been tuned. Please tune your model for
better performance. Use DEBUG logging level to see more details.
```

I looked a bit more into it - it wasn't intuitive at all to me that the model that's supposed to be "tuned for better performance" is still performing better compared to the model that didn't get the same message!

I researched it online, but couldn't find a specific solution that made sense. As a last resort, I decided to ask Gemini (Google's chatbot) and among other things that it mentioned, I came across this paragraph that I found interesting:

It's possible that the lack of tuning in the unquantized model leads to a simpler execution path. Without optimizations, the model might be using a more generic implementation that, in some cases, could be surprisingly efficient for the specific hardware you're running on.

This, paired with possible overhead and hardware issues, made sense to me.

How did TVM know that I wanted to quantize to int8?

In your quantization setup, how did TVM know that you wanted to quantize to int8? Look into that, and vary the number of bits of quantization (the in int-). Searching in forum and peeking the source code of the quantizer class will both help.

The `qconfig` function allows us to define the quantization configuration using the `QConfig` class. This class has attributes like dtype_input and dtype_weight, which specify the number of bits and data type for different elements (input, weight) in the model.

When quantizing the DNN in my code, I specifically set the configuration so that dtype_input and dtype_weight were equal to "int8".

```
with relay.quantize.qconfig(calibrate_mode='global_scale',
                     dtype_input="int8",
                     dtype_weight="int8",
                     dtype_activation="int8",
                     global_scale=8.0):
  mod = relay.quantize.quantize(mod, params)
```

Upon taking a closer look at the source code of the `quantize` class (https://github.com/apache/tvm/blob/main/python/tvm/relay/quantize/quantize.py), I came across some more interesting discoveries.

While qconfig allows us to specify the number of bits, it also provides defaults. By default, nbit_input and nbit_weight are both set to 8, corresponding to int8. dtype_input and dtype_weight are also set to "int8" by default.

So to sum up: while we can influence the quantization type through qconfig, TVM's defaults and calibration process are geared towards using int8 for efficient low-precision inference. That's how TVM will know that we want to quantize to "int8", even if we don't specifically set the configuration to that.

Try out int8 -> int4 -> int2 -> int1; note which precisions work. When it doesn't work, note exactly which part is failing.

In [20]:

```
keys = ["int8", "int4", "int2", "bool"]
for key in keys:
    try:
        lib = f_quantize(key)
        m = graph_executor.GraphModule(lib["default"](dev))
        m.set_input("input0", tvm.nd.array(img.astype(dtype)))
    except Exception as e:
        print(key + " is failing!")
```

```
int4 is failing!
int2 is failing!
```

We can see from the output that `int4` and `int2` are failing. Before setting up the try-catch block we see here, I tried compiling them one by one and the result I received was "TVMError: unknown data type int4" (as seen below). This makes me think that they are probably not supported by the TMV compiler.

```
        at /home/mila/Desktop/uiuc/tvm/include/tvm/relay/expr_functor.h:95
  6: tvm::relay::ExprFunctor<tvm::RelayExpr (tvm::RelayExpr const&)>::InitVTable()::{lambda(tvm::runtime::ObjectRe
f const&, tvm::relay::ExprFunctor<tvm::RelayExpr (tvm::RelayExpr const&)>*)#6}::_FUN(tvm::runtime::ObjectRef const
&, tvm::relay::ExprFunctor<tvm::RelayExpr (tvm::RelayExpr const&)>*)
        at /home/mila/Desktop/uiuc/tvm/include/tvm/relay/expr_functor.h:128
  5: tvm::relay::ExprFunctor<tvm::RelayExpr (tvm::RelayExpr const&)>::InitVTable()::{lambda(tvm::runtime::ObjectRe
f const&, tvm::relay::ExprFunctor<tvm::RelayExpr (tvm::RelayExpr const&)>*)#6}::operator()(tvm::runtime::ObjectRef
const&, tvm::relay::ExprFunctor<tvm::RelayExpr (tvm::RelayExpr const&)>*) const
        at /home/mila/Desktop/uiuc/tvm/include/tvm/relay/expr_functor.h:128
  4: tvm::relay::MixedModeMutator::VisitExpr_(tvm::relay::CallNode const*)
        at /home/mila/Desktop/uiuc/tvm/include/tvm/relay/expr_functor.h:291
  3: tvm::RelayExpr tvm::relay::MixedModeMutator::Rewrite<tvm::relay::CallNode>(tvm::relay::CallNode const*)
        at /home/mila/Desktop/uiuc/tvm/include/tvm/relay/expr_functor.h:313
  2: tvm::relay::ForwardRewriter::Rewrite_(tvm::relay::CallNode const*, tvm::RelayExpr const&)
        at /home/mila/Desktop/uiuc/tvm/src/relay/transforms/forward_rewrite.cc:155
  1: tvm::relay::quantize::QuantizeRealize(tvm::relay::Call const&, tvm::runtime::Array<tvm::RelayExpr, void> cons
t&, tvm::runtime::ObjectRef const&)
        at /home/mila/Desktop/uiuc/tvm/src/relay/quantize/realize.cc:126
  0: tvm::relay::Constant tvm::relay::MakeConstantScalar<int>(tvm::runtime::DataType, int)
        at /home/mila/Desktop/uiuc/tvm/src/relay/quantize/../qnn/./op/../../op/../transforms/pattern_utils.h:290
  File "/home/mila/Desktop/uiuc/tvm/src/relay/quantize/../qnn/./op/../../op/../transforms/pattern_utils.h", line 2
90
TVMError: unknown data type int4
```