Note

Click here to download the full example code

Deploy a Framework-prequantized Model with TVM

Author: Masahiro Masuda

This is a tutorial on loading models quantized by deep learning frameworks into TVM. Pre-quantized model import is one of the quantization support we have in TVM. More details on the quantization story in TVM can be found here.

Here, we demonstrate how to load and run models quantized by PyTorch, MXNet, and TFLite. Once loaded, we can run compiled, quantized models on any hardware TVM supports.

First, necessary imports

```
from PIL import Image

import numpy as np

import torch
from torchvision.models.quantization import mobilenet as qmobilenet

import tvm
from tvm import relay
from tvm.contrib.download import download_testdata
```

Helper functions to run the demo

```
def get_transform():
    import torchvision.transforms as transforms
    normalize = transforms.Normalize(mean=[0.485, 0.456, 0.406],
                                     std=[0.229, 0.224, 0.225])
    return transforms.Compose([
            transforms.Resize(256),
            transforms.CenterCrop(224),
            transforms.ToTensor(),
            normalize,
        ])


def get_real_image(im_height, im_width):
    img_url = 'https://github.com/dmlc/mxnet.js/blob/master/data/cat.png?raw=true'
    img_path = download_testdata(img_url, 'cat.png', module='data')
    return Image.open(img_path).resize((im_height, im_width))


def get_imagenet_input():
    im = get_real_image(224, 224)
    preprocess = get_transform()
    pt_tensor = preprocess(im)
    return np.expand_dims(pt_tensor.numpy(), 0)


def get_synset():
    synset_url = ''.join(['https://gist.githubusercontent.com/zhreshold/',
                          '4d0b62f3d01426887599d4f7ede23ee5/raw/',
                          '596b27d23537e5a1b5751d2b0481ef172f58b539/',
                          'imagenet1000_clsid_to_human.txt'])
    synset_name = 'imagenet1000_clsid_to_human.txt'
    synset_path = download_testdata(synset_url, synset_name, module='data')
    with open(synset_path) as f:
        return eval(f.read())


def run_tvm_model(mod, params, input_name, inp, target="llvm"):
    with relay.build_config(opt_level=3):
        json, lib, params = relay.build(mod, target=target, params=params)

    runtime = tvm.contrib.graph_runtime.create(json, lib, tvm.context(target, 0))
    runtime.set_input(**params)

    runtime.set_input(input_name, inp)
    runtime.run()
    return runtime.get_output(0).asnumpy(), runtime
```

A mapping from label to class name, to verify that the outputs from models below are reasonable

```
synset = get_synset()
```

Out:

```
File /home/masa/.tvm_test_data/data/imagenet1000_clsid_to_human.txt exists, skip.
```

Everyone’s favorite cat image for demonstration

```
inp = get_imagenet_input()
```

Out:

```
File /home/masa/.tvm_test_data/data/cat.png exists, skip.
```

Deploy a quantized PyTorch Model

First, we demonstrate how to load deep learning models quantized by PyTorch, using our PyTorch frontend.

Please refer to the PyTorch static quantization tutorial below to learn about their quantization workflow. https://pytorch.org/tutorials/advanced/static_quantization_tutorial.html

We use this function to quantize PyTorch models. In short, this function takes a floating point model and converts it to uint8. The model is per-channel quantized.

```
def quantize_model(model, inp):
    model.fuse_model()
    model.qconfig = torch.quantization.get_default_qconfig('fbgemm')
    torch.quantization.prepare(model, inplace=True)
    # Dummy calibration
    model(inp)
    torch.quantization.convert(model, inplace=True)
```

Load quantization-ready, pretrained Mobilenet v2 model from torchvision

We choose mobilenet v2 because this model was trained with quantization aware training. Other models require a full post training calibration.

```
qmodel = qmobilenet.mobilenet_v2(pretrained=True).eval()
```

Quantize, trace and run the PyTorch Mobilenet v2 model

The details are out of scope for this tutorial. Please refer to the tutorials on the PyTorch website to learn about quantization and jit.

```
pt_inp = torch.from_numpy(inp)
quantize_model(qmodel, pt_inp)
script_module = torch.jit.trace(qmodel, pt_inp).eval()

with torch.no_grad():
    pt_result = script_module(pt_inp).numpy()
```

Out:

```
/home/masa/projects/deep/pytorch/torch/quantization/observer.py:845: UserWarning: must
run observer before calling calculate_qparams.
Returning default scale and zero point
  Returning default scale and zero point "
```

Convert quantized Mobilenet v2 to Relay-QNN using the PyTorch frontend

The PyTorch frontend has support for converting a quantized PyTorch model to an equivalent Relay module enriched with quantization-aware operators. We call this representation Relay QNN dialect.

You can print the output from the frontend to see how quantized models are represented.

You would see operators specific to quantization such as qnn.quantize, qnn.dequantize, qnn.requantize, and qnn.conv2d etc.

```
input_name = "input"  # the input name can be be arbitrary for PyTorch frontend.
input_shapes = [(input_name, (1, 3, 224, 224))]
mod, params = relay.frontend.from_pytorch(script_module, input_shapes)
# print(mod)
```

Out:

```
ANTLR runtime and generated code versions disagree: 4.8!=4.7.2
ANTLR runtime and generated code versions disagree: 4.8!=4.7.2
```

Compile and run the Relay module

Once we obtained the quantized Relay module, the rest of the workflow is the same as running floating point models. Please refer to other tutorials for more details.

Under the hood, quantization specific operators are lowered to a sequence of standard Relay operators before compilation.

```
tvm_result, rt_mod = run_tvm_model(mod, params, input_name, inp, target="llvm")
```

Compare the output labels

We should see identical labels printed.

```
pt_top3_labels = np.argsort(pt_result[0])[::-1][:3]
tvm_top3_labels = np.argsort(tvm_result[0])[::-1][:3]

print("PyTorch top3 label:", [synset[label] for label in pt_top3_labels])
print("TVM top3 label:", [synset[label] for label in tvm_top3_labels])
```

Out:

```
PyTorch top3 label: ['tiger cat', 'Egyptian cat', 'weasel']
TVM top3 label: ['tiger cat', 'Egyptian cat', 'weasel']
```

However, due to the difference in numerics, in general the raw floating point outputs are not expected to be identical. Here, we print how many floating point output values are identical out of 1000 outputs from mobilenet v2.

```
print("%d in 1000 raw floating outputs identical." % np.sum(tvm_result[0] ==
pt_result[0]))
```

Out:

```
189 in 1000 raw floating outputs identical.
```

Measure performance

Here we give an example of how to measure performance of TVM compiled models.

```
n_repeat = 100  # should be bigger to make the measurement more accurate
ctx = tvm.cpu(0)
ftimer = rt_mod.module.time_evaluator("run", ctx, number=1,
                                      repeat=n_repeat)
prof_res = np.array(ftimer().results) * 1e3
print("Elapsed ms:", np.mean(prof_res))
```

Out:

```
Elapsed ms: 31.18784119
```

Note

We recommend this method for the following reasons:

- Measurements are done in C++, so there is no Python overhead
- It includes several warm up runs
- The same method can be used to profile on remote devices (android etc.).

Note

Unless the hardware has special support for fast 8 bit instructions, quantized models are not expected to be any faster than FP32 models. Without fast 8 bit instructions, TVM does quantized convolution in 16 bit, even if the model itself is 8 bit.

For x86, the best performance can be acheived on CPUs with AVX512 instructions set. In this case, TVM utilizes the fastest available 8 bit instructions for the given target. This includes support for the VNNI 8 bit dot product instruction (CascadeLake or newer).

Moreover, the following general tips for CPU performance equally applies:

- Set the environment variable TVM_NUM_THREADS to the number of physical cores
- Choose the best target for your hardware, such as “llvm -mcpu=skylake-avx512” or “llvm -mcpu=cascadelake” (more CPUs with AVX512 would come in the future)

Deploy a quantized MXNet Model

TODO

Deploy a quantized TFLite Model

TODO

Download Python source code: deploy_prequantized.py

Download Jupyter notebook: deploy_prequantized.ipynb