QNNPACK

QNNPACK (Quantized Neural Networks PACKage) is a mobile-optimized library for low-precision high-performance neural network inference. QNNPACK provides implementation of common neural network operators on quantized 8-bit tensors.

QNNPACK is not intended to be directly used by machine learning researchers; instead it provides low-level performance primitives for high-level deep learning frameworks. As of today, QNNPACK is integrated in PyTorch 1.0 with Caffe2 graph representation.

Operator Coverage

Currently implemented and planned for implementation operators are below:

- ☒ 2D Convolution
- ☒ 2D Deconvolution
- ☒ Channel Shuffle
- ☒ Fully Connected
- ☐ Locally Connected
- ☒ 2D Max Pooling
- ☒ 2D Average Pooling
- ☒ Global Average Pooling
- ☒ Sigmoid
- ☒ TanH
- ☒ Leaky ReLU
- ☒ Hardsigmoid
- ☒ Hardswish
- ☒ Clamp (can be used for ReLU, ReLU6 if it is not fused in another operator)
- ☒ SoftArgMax (aka SoftMax)
- ☐ Group Normalization

Building

QNNPACK provides standard CMake-based build scripts.

Native compilation

Users are recommended to use `scripts/build-local.sh` script to build QNNPACK for the host machine.

Cross-compilation for Android

To cross-compile for Android, set `$ANDROID_NDK` environment variable (where `$ANDROID_NDK` is the path to Android NDK directory, e.g. `/opt/android-ndk-r15c`) and use one of the scripts from the table below:

ABI	Build script	Restrictions
armeabi-v7a	`scripts/build-android-armv7.sh`	Requires CPU with ARM NEON
arm64-v8a	`scripts/build-android-arm64.sh`	
x86	`scripts/build-android-x86.sh`	

Notes: - On **armeabi-v7a** `pytorch_qnnp_initialize` will fail with `pytorch_qnnp_status_unsupported_hardware` if the mobile CPU does not support ARM NEON. Don't set `-DANDROID_ARM_NEON=1` for QNNPACK compilation as it can make `pytorch_qnnp_initialize` crash on CPUs without ARM NEON.

Cross-compilation for iOS

To cross-compile for iOS, clone ios-cmake, and set `$IOS_CMAKE_TOOLCHAIN_FILE` environment variable (where `$IOS_CMAKE_TOOLCHAIN_FILE` is the path to `ios.toolchain.cmake` file in ios-cmake), and use one of the scripts from the table below:

Architecture	Build script	Notes
armv7	`scripts/build-ios-armv7.sh`	iPhone 3GS/4/4S
armv7	`scripts/build-ios-armv7s.sh`	iPhone 5 and newer
arm64	`scripts/build-ios-arm64.sh`	iPhone 5S and newer
arm64e	`scripts/build-ios-arm64e.sh`	iPhone XS/XR
i386	`scripts/build-ios-i386.sh`	iPhone Simulator (32-bit)
x86_64	`scripts/build-ios-x86_64.sh`	iPhone Simulator (64-bit)

End-to-End Benchmarking

Caffe2 backend of PyTorch 1.0 natively integrates QNNPACK, and provides a pre-trained quantized MobileNet v2 model. Below are instructions for benchmarking this model end-to-end with QNNPACK.

Raspberry Pi 2 or 3

```
# Clone PyTorch 1.0 repo
git clone --recursive https://github.com/pytorch/pytorch.git
cd pytorch

# Optional: update QNNPACK submodule to latest revision
git submodule update --remote --jobs 0 third_party/QNNPACK

# Build Caffe2 (including binaries) for the host system
# Use only 1 thread for build to avoid out-of-memory failures
```

```
MAX_JOBS=1 scripts/build_local.sh -DBUILD_BINARY=ON -DBUILD_PYTHON=OFF \
    -DUSE_OBSERVERS=OFF -DUSE_DISTRIBUTED=OFF

# Download model weights
wget https://s3.amazonaws.com/download.caffe2.ai/models/mobilenet_v2_1.0_224_quant/init_net.

# Download model graph
wget https://s3.amazonaws.com/download.caffe2.ai/models/mobilenet_v2_1.0_224_quant/predict_n

# Run speed benchmark with 50 warm-up iterations and 10 measurement iterations
build/bin/speed_benchmark --net predict_net.pb --init_net init_net.pb \
    --input data --input_dims 1,3,224,224 --input_type float \
    --warmup 50 --iter 10
```

ARMv7 (32-bit) Android

```
# Clone PyTorch 1.0 repo
git clone --recursive https://github.com/pytorch/pytorch.git
cd pytorch

# Optional: update QNNPACK submodule to latest revision
git submodule update --remote --jobs 0 third_party/QNNPACK

# Build Caffe2 (including binaries) for Android, and push to device
scripts/build_android.sh -DANDROID_TOOLCHAIN=clang -DBUILD_BINARY=ON
adb push build_android/bin/speed_benchmark /data/local/tmp/speed_benchmark

# Download model weights and copy them to Android device
wget https://s3.amazonaws.com/download.caffe2.ai/models/mobilenet_v2_1.0_224_quant/init_net.
adb push init_net.pb /data/local/tmp/init_net.pb

# Download model graph and copy it to Android device
wget https://s3.amazonaws.com/download.caffe2.ai/models/mobilenet_v2_1.0_224_quant/predict_n
adb push predict_net.pb /data/local/tmp/predict_net.pb

# Run speed benchmark with 50 warm-up iterations and 10 measurement iterations
adb shell /data/local/tmp/speed_benchmark \
    --net /data/local/tmp/predict_net.pb \
    --init_net /data/local/tmp/init_net.pb \
    --input data --input_dims 1,3,224,224 --input_type float \
    --warmup 50 --iter 10
```

ARM64 (64-bit) Android

```
# Clone PyTorch 1.0 repo
git clone --recursive https://github.com/pytorch/pytorch.git
```

```
cd pytorch

# Optional: update QNNPACK submodule to latest revision
git submodule update --remote --jobs 0 third_party/QNNPACK

# Build Caffe2 (including binaries) for Android, and push to device
scripts/build_android.sh -DANDROID_ABI=arm64-v8a -DANDROID_TOOLCHAIN=clang -DBUILD_BINARY=ON
adb push build_android/bin/speed_benchmark /data/local/tmp/speed_benchmark

# Download model weights and copy them to Android device
wget https://s3.amazonaws.com/download.caffe2.ai/models/mobilenet_v2_1.0_224_quant/init_net.
adb push init_net.pb /data/local/tmp/init_net.pb

# Download model graph and copy it to Android device
wget https://s3.amazonaws.com/download.caffe2.ai/models/mobilenet_v2_1.0_224_quant/predict_n
adb push predict_net.pb /data/local/tmp/predict_net.pb

# Run speed benchmark with 50 warm-up iterations and 10 measurement iterations
adb shell /data/local/tmp/speed_benchmark \
    --net /data/local/tmp/predict_net.pb \
    --init_net /data/local/tmp/init_net.pb \
    --input data --input_dims 1,3,224,224 --input_type float \
    --warmup 50 --iter 10
```

PEP (Performance Evaluation Platform) Method

Facebook AI Performance Evaluation Platform is a framework and backend agnostic benchmarking platform to compare machine learning inferencing runtime metrics on a set of models and a variety of backends.

We use PEP to produce the results we have in our blog

With an ARMv7 device connected:

```
# Clone PyTorch 1.0 repo
mkdir ~/Code && cd ~/Code
git clone --recursive https://github.com/pytorch/pytorch.git
cd pytorch

# Optional: update QNNPACK submodule to latest revision
git submodule update --remote --jobs 0 third_party/QNNPACK

# Clone PEP repo
cd ~/Code
git clone --recursive https://github.com/facebook/FAI-PEP.git aibench
cd aibench
```

```
# Run PEP benchmark with cool specifications. Try changing that cmd with more specifications
# First time compile could take 20+ minutes
./benchmarking/run_bench.py \
  --platform android \
  -b ~/Code/aibench/specifications/models/caffe2/mobilenet_v2/mobilenet_v2_quant.json \
  --platform android --repo_dir ~/Code/pytorch \
  --frameworks_dir ~/Code/aibench/specifications/frameworks --framework caffe2
```

Acknowledgements

QNNPACK is developed by Marat Dukhan, Yiming Wu, Hao Lu, and Bert Maher. We thank Andrew Tulloch and Yangqing Jia for advice during the development of QNNPACK.

License

QNNPACK is BSD licensed, as found in the `LICENSE` file.