Android

Demo applications and tutorials

Demo applications with code walk-through can be find in this github repo.

Publishing

Release Release artifacts are published to jcenter:

```
repositories {
    jcenter()
}

# lite interpreter build
dependencies {
    implementation 'org.pytorch:pytorch_android_lite:1.10.0'
    implementation 'org.pytorch:pytorch_android_torchvision_lite:1.10.0'
}

# full jit build
dependencies {
    implementation 'org.pytorch:pytorch_android:1.10.0'
    implementation 'org.pytorch:pytorch_android_torchvision:1.10.0'
}
```

Nightly Nightly(snapshots) builds are published every night from `master` branch to nexus sonatype snapshots repository

To use them repository must be specified explicitly:

```
repositories {
    maven {
        url "https://oss.sonatype.org/content/repositories/snapshots"
    }
}

# lite interpreter build
dependencies {
    ...
    implementation 'org.pytorch:pytorch_android_lite:1.12.0-SNAPSHOT'
    implementation 'org.pytorch:pytorch_android_torchvision_lite:1.12.0-SNAPSHOT'
    ...
}

# full jit build
dependencies {
    ...
```

```
    implementation 'org.pytorch:pytorch_android:1.12.0-SNAPSHOT'
    implementation 'org.pytorch:pytorch_android_torchvision:1.12.0-SNAPSHOT'
    ...
}
```

The current nightly(snapshots) version is the value of `VERSION_NAME` in `gradle.properties` in current folder, at this moment it is `1.8.0-SNAPSHOT`.

Building PyTorch Android from Source

In some cases you might want to use a local build of pytorch android, for example you may build custom libtorch binary with another set of operators or to make local changes.

For this you can use `./scripts/build_pytorch_android.sh` script.

```
git clone https://github.com/pytorch/pytorch.git
cd pytorch
git submodule update --init --recursive --jobs 0
sh ./scripts/build_pytorch_android.sh
```

The workflow contains several steps:

1. Build libtorch for android for all 4 android abis (armeabi-v7a, arm64-v8a, x86, x86_64)

2. Create symbolic links to the results of those builds: `android/pytorch_android/src/main/jniLibs/${abi}` to the directory with output libraries `android/pytorch_android/src/main/cpp/libtorch_include/${abi}` to the directory with headers. These directories are used to build `libpytorch.so` library that will be loaded on android device.

3. And finally run `gradle` in `android/pytorch_android` directory with task `assembleRelease`

Script requires that Android SDK, Android NDK and gradle are installed. They are specified as environment variables:

`ANDROID_HOME` - path to Android SDK

`ANDROID_NDK` - path to Android NDK. It's recommended to use NDK 21.x.

`GRADLE_HOME` - path to gradle

After successful build you should see the result as aar file:

```
$ find pytorch_android/build/ -type f -name *aar
pytorch_android/build/outputs/aar/pytorch_android.aar
pytorch_android_torchvision/build/outputs/aar/pytorch_android.aar
```

It can be used directly in android projects, as a gradle dependency:

```
allprojects {
    repositories {
```

```
        flatDir {
            dirs 'libs'
        }
    }
}

dependencies {
    implementation(name:'pytorch_android', ext:'aar')
    implementation(name:'pytorch_android_torchvision', ext:'aar')
    ...
    implementation 'com.facebook.soloader:nativeloader:0.10.1'
    implementation 'com.facebook.fbjni:fbjni-java-only:0.2.2'
}
```

We also have to add all transitive dependencies of our aars. As `pytorch_android` depends on `'com.facebook.soloader:nativeloader:0.10.1'` and `'com.facebook.fbjni:fbjni-java-only` we need to add them. (In case of using maven dependencies they are added automatically from `pom.xml`).

You can check out test app example that uses aars directly.

Linking to prebuilt libtorch library from gradle dependency

In some cases, you may want to use libtorch from your android native build. You can do it without building libtorch android, using native libraries from PyTorch android gradle dependency. For that, you will need to add the next lines to your gradle build.

```
android {
...
    configurations {
       extractForNativeBuild
    }
...
    compileOptions {
        externalNativeBuild {
            cmake {
                arguments "-DANDROID_STL=c++_shared"
            }
        }
    }
...
    externalNativeBuild {
        cmake {
            path "CMakeLists.txt"
        }
    }
```

```
}

dependencies {
    extractForNativeBuild('org.pytorch:pytorch_android:1.10.0')
}

task extractAARForNativeBuild {
    doLast {
        configurations.extractForNativeBuild.files.each {
            def file = it.absoluteFile
            copy {
                from zipTree(file)
                into "$buildDir/$file.name"
                include "headers/**"
                include "jni/**"
            }
        }
    }
}

tasks.whenTaskAdded { task ->
  if (task.name.contains('externalNativeBuild')) {
    task.dependsOn(extractAARForNativeBuild)
  }
}
```

pytorch_android aar contains headers to link in `headers` folder and native libraries in `jni/$ANDROID_ABI/`. As PyTorch native libraries use `ANDROID_STL` - we should use `ANDROID_STL=c++_shared` to have only one loaded binary of STL.

The added task will unpack them to gradle build directory.

In your native build you can link to them adding these lines to your CMakeLists.txt:

```
# Relative path of gradle build directory to CMakeLists.txt
set(build_DIR ${CMAKE_SOURCE_DIR}/build)

file(GLOB PYTORCH_INCLUDE_DIRS "${build_DIR}/pytorch_android*.aar/headers")
file(GLOB PYTORCH_LINK_DIRS "${build_DIR}/pytorch_android*.aar/jni/${ANDROID_ABI}")

set(BUILD_SUBDIR ${ANDROID_ABI})
target_include_directories(${PROJECT_NAME} PRIVATE
  ${PYTORCH_INCLUDE_DIRS}
)

find_library(PYTORCH_LIBRARY pytorch_jni
  PATHS ${PYTORCH_LINK_DIRS}
```

```
  NO_CMAKE_FIND_ROOT_PATH)

find_library(FBJNI_LIBRARY fbjni
  PATHS ${PYTORCH_LINK_DIRS}
  NO_CMAKE_FIND_ROOT_PATH)

target_link_libraries(${PROJECT_NAME}
  ${PYTORCH_LIBRARY})
  ${FBJNI_LIBRARY})
```

If your CMakeLists.txt file is located in the same directory as your build.gradle, `set(build_DIR ${CMAKE_SOURCE_DIR}/build)` should work for you. But if you have another location of it, you may need to change it.

After that, you can use libtorch C++ API from your native code.

```
#include <string>
#include <ATen/NativeFunctions.h>
#include <torch/script.h>
namespace pytorch_testapp_jni {
namespace {
    struct JITCallGuard {
      c10::InferenceMode guard;
      torch::jit::GraphOptimizerEnabledGuard no_optimizer_guard{false};
    };
}

void loadAndForwardModel(const std::string& modelPath) {
  JITCallGuard guard;
  torch::jit::Module module = torch::jit::load(modelPath);
  module.eval();
  torch::Tensor t = torch::randn({1, 3, 224, 224});
  c10::IValue t_out = module.forward({t});
}
}
```

To load torchscript model for mobile we need some special setup which is placed in `struct JITCallGuard` in this example. It may change in future, you can track the latest changes keeping an eye in our pytorch android jni code

Example of linking to libtorch from aar

PyTorch Android API Javadoc

You can find more details about the PyTorch Android API in the Javadoc.