Installing C++ Distributions of PyTorch

We provide binary distributions of all headers, libraries and CMake configuration files required to depend on PyTorch. We call this distribution *LibTorch*, and you can download ZIP archives containing the latest LibTorch distribution on our website. Below is a small example of writing a minimal application that depends on LibTorch and uses the `torch::Tensor` class which comes with the PyTorch C++ API.

Minimal Example

The first step is to download the LibTorch ZIP archive via the link above. For example:

```
wget https://download.pytorch.org/libtorch/nightly/cpu/libtorch-shared-with-deps-latest.zip
unzip libtorch-shared-with-deps-latest.zip
```

Note that the above link has CPU-only libtorch. If you would like to download a GPU-enabled libtorch, find the right link in the link selector on https://pytorch.org

If you're a Windows developer and wouldn't like to use CMake, you could jump to the Visual Studio Extension section.

Next, we can write a minimal CMake build configuration to develop a small application that depends on LibTorch. CMake is not a hard requirement for using LibTorch, but it is the recommended and blessed build system and will be well supported into the future. A most basic *CMakeLists.txt* file could look like this:

```
cmake_minimum_required(VERSION 3.0 FATAL_ERROR)
project(example-app)

find_package(Torch REQUIRED)
set(CMAKE_CXX_FLAGS "${CMAKE_CXX_FLAGS} ${TORCH_CXX_FLAGS}")

add_executable(example-app example-app.cpp)
target_link_libraries(example-app "${TORCH_LIBRARIES}")
set_property(TARGET example-app PROPERTY CXX_STANDARD 14)

# The following code block is suggested to be used on Windows.
# According to https://github.com/pytorch/pytorch/issues/25457,
# the DLLs need to be copied to avoid memory errors.
if (MSVC)
  file(GLOB TORCH_DLLS "${TORCH_INSTALL_PREFIX}/lib/*.dll")
  add_custom_command(TARGET example-app
                     POST_BUILD
                     COMMAND ${CMAKE_COMMAND} -E copy_if_different
                     ${TORCH_DLLS}
                     $<TARGET_FILE_DIR:example-app>)
endif (MSVC)
```

The implementation of our example will simply create a new *torch::Tensor* and print it:

```
#include <torch/torch.h>
#include <iostream>

int main() {
  torch::Tensor tensor = torch::rand({2, 3});
  std::cout << tensor << std::endl;
}
```

While there are more fine-grained headers you can include to access only parts of the PyTorch C++ API, including *torch/torch.h* is the most sure-proof way of including most of its functionality.

The last step is to build the application. For this, assume our example directory is laid out like this:

```
example-app/
  CMakeLists.txt
  example-app.cpp
```

We can now run the following commands to build the application from within the `example-app/` folder:

```
mkdir build
cd build
cmake -DCMAKE_PREFIX_PATH=/absolute/path/to/libtorch ..
cmake --build . --config Release
```

where `/absolute/path/to/libtorch` should be the absolute (!) path to the unzipped LibTorch distribution. If PyTorch was installed via conda or pip, *CMAKE_PREFIX_PATH* can be queried using *torch.utils.cmake_prefix_path* variable. In that case CMake configuration step would look something like follows:

```
cmake -DCMAKE_PREFIX_PATH=`python -c 'import torch;print(torch.utils.cmake_prefix_path)'`
```

If all goes well, it will look something like this:

```
root@4b5a67132e81:/example-app# mkdir build
root@4b5a67132e81:/example-app# cd build
root@4b5a67132e81:/example-app/build# cmake -DCMAKE_PREFIX_PATH=/path/to/libtorch ..
-- The C compiler identification is GNU 5.4.0
-- The CXX compiler identification is GNU 5.4.0
-- Check for working C compiler: /usr/bin/cc
-- Check for working C compiler: /usr/bin/cc -- works
-- Detecting C compiler ABI info
-- Detecting C compiler ABI info - done
-- Detecting C compile features
-- Detecting C compile features - done
-- Check for working CXX compiler: /usr/bin/c++
-- Check for working CXX compiler: /usr/bin/c++ -- works
-- Detecting CXX compiler ABI info
-- Detecting CXX compiler ABI info - done
-- Detecting CXX compile features
-- Detecting CXX compile features - done
-- Looking for pthread.h
-- Looking for pthread.h - found
-- Looking for pthread_create
-- Looking for pthread_create - not found
-- Looking for pthread_create in pthreads
-- Looking for pthread_create in pthreads - not found
-- Looking for pthread_create in pthread
-- Looking for pthread_create in pthread - found
-- Found Threads: TRUE
-- Configuring done
-- Generating done
-- Build files have been written to: /example-app/build
root@4b5a67132e81:/example-app/build# cmake --build . --config Release
Scanning dependencies of target example-app
[ 50%] Building CXX object CMakeFiles/example-app.dir/example-app.cpp.o
[100%] Linking CXX executable example-app
[100%] Built target example-app
```

Executing the resulting `example-app` binary found in the `build` folder should now merrily print the tensor (exact output subject to randomness):

```
root@4b5a67132e81:/example-app/build# ./example-app
0.2063  0.6593  0.0866
0.0796  0.5841  0.1569
[ Variable[CPUFloatType]{2,3} ]
```

Tip

On Windows, debug and release builds are not ABI-compatible. If you plan to build your project in debug mode, please try the debug version of LibTorch. Also, make sure you specify the correct configuration in the `cmake --build .` line above.

Visual Studio Extension

LibTorch Project Template can help Windows developers set all libtorch project settings and link options for debug and release. It's easy to use and you could check out the demo video. The only prerequisite is to download the libtorch on https://pytorch.org

Support

If you run into any troubles with this installation and minimal usage guide, please use our forum or GitHub issues to get in touch.