plsdlr /
tensorflowliteonpizero

Code Issues 1 Pull requests 1 Actions Projects Security

TensorFlow Lite : a comprehensive guide to cross compilation and building python bindings of TensorFlow Lite for Raspberry Pi Zero W

4 stars **0** forks **3** watching Activity

Public repository

main

Branches Tags

plsdlr Update README.md ... on Nov 26, 2020 **10**

View code

README.md

Tensorflow lite on Raspberry Pi Zero armv6 - a comprehensive guide

This guide is written to help with crosscompilation and building Piwheels for tensorflow lite on Raspberry Pi Zero. The official tensorflow documentation seem to be out of date and also dosen't document how to build a working crosscompilation toolchain. While libtensorflow-lite.a is available for armv6 , the python wrapper to it, i.e. tflite_runtime existed to our knowlege only for armv7 architectures. This guide shows therefore how to build libtensorflow-lite.a and the python wrapper for for armv6 achitecture.

One warning: the whole process is quite time intensive. It was tested on **Ubuntu 18.04.5 LTS** and **Raspbian GNU/Linux 10 (buster)**. This repository also contains the precompiled *libtensorflow-lite.a* and *minimal* and the precompiled piwheel. If you are lucky the piwheel might work out of the box. (jump to step 4)

Therefore this guide is seperated in four parts:

0. Preparation: Building the arm-linux-gnueabihf-g++ toolchain
1. Compiling libtensorflow-lite.a and minimal
2. Building bindings for tensorflow lite
3. Installing Wheels or using the precompiled Wheels
4. Loading a model and testing tflite_runtime

If you want to skip the compilation:

The precompiled whl: tflite_runtime-2.5.0-cp37-cp37m-linux_armv6l.whl

The precompiled lib: libtensorflow-lite.a

The precompiled minimalexample: minimal

Preparation: Building arm-linux-gnueabihf-g++

This part of the guide is completely taken from here. Give it a star if you use this.

Sources:

```
http://preshing.com/20141119/how-to-build-a-gcc-cross-compiler/
https://www.raspberrypi.org/documentation/linux/kernel/building.md


https://wiki.osdev.org/Why_do_I_need_a_Cross_Compiler%3F
https://wiki.osdev.org/GCC_Cross-Compiler
https://wiki.osdev.org/Building_GCC
http://www.ifp.illinois.edu/~nakazato/tips/xgcc.html
```

1. Update system

```
sudo apt update
sudo apt upgrade
```

2. Install prereq:

```
sudo apt install build-essential gawk git texinfo bison
```

3. Download software:

```
wget https://ftpmirror.gnu.org/binutils/binutils-2.30.tar.bz2
wget https://ftpmirror.gnu.org/gcc/gcc-8.1.0/gcc-8.1.0.tar.gz
wget https://ftpmirror.gnu.org/glibc/glibc-2.27.tar.bz2
git clone --depth=1 https://github.com/raspberrypi/linux
```

4. Extract archives and remove them

```
tar xf binutils-2.30.tar.bz2
tar xf glibc-2.27.tar.bz2
tar xf gcc-8.1.0.tar.gz
rm *.tar.*
```

5. Get GCC prereq

```
cd gcc-*
contrib/download_prerequisites
rm *.tar.*
cd ..
```

6. Create the folder in which we'll put the cross compiler and add it to the PATH

```
sudo mkdir -p /opt/cross-pi-gcc
sudo chown $USER /opt/cross-pi-gcc
export PATH=/opt/cross-pi-gcc/bin:$PATH
```

7. Build binutils

```
mkdir build-binutils && cd build-binutils
../binutils-2.30/configure --prefix=/opt/cross-pi-gcc --target=arm-
linux-gnueabihf --with-arch=armv6 --with-fpu=vfp --with-float=hard
--disable-multilib
make -j 8
make install
```

8. Install kernel headers

```
# See also https://www.raspberrypi.org/documentation/linux/kernel
/building.md

cd linux
KERNEL=kernel7
make ARCH=arm INSTALL_HDR_PATH=/opt/cross-pi-gcc/arm-linux-gnueabihf
headers_install
```

9. Build compilers

```
cd ..
mkdir build-gcc && cd build-gcc
../gcc-8.1.0/configure --prefix=/opt/cross-pi-gcc --target=arm-linux-
gnueabihf --enable-languages=c,c++,fortran --with-arch=armv6 --with-
fpu=vfp --with-float=hard --disable-multilib
make -j8 all-gcc
make install-gcc
```

10. Partially build glibc

```
cd ..
mkdir build-glibc && cd build-glibc
../glibc-2.27/configure --prefix=/opt/cross-pi-gcc/arm-linux-
gnueabihf --build=$MACHTYPE --host=arm-linux-gnueabihf --target=arm-
linux-gnueabihf --with-arch=armv6 --with-fpu=vfp --with-float=hard
--with-headers=/opt/cross-pi-gcc/arm-linux-gnueabihf/include
--disable-multilib libc_cv_forced_unwind=yes
make install-bootstrap-headers=yes install-headers
make -j8 csu/subdir_lib
install csu/crt1.o csu/crti.o csu/crtn.o /opt/cross-pi-gcc/arm-linux-
gnueabihf/lib
arm-linux-gnueabihf-gcc -nostdlib -nostartfiles -shared -x c
/dev/null -o /opt/cross-pi-gcc/arm-linux-gnueabihf/lib/libc.so
touch /opt/cross-pi-gcc/arm-linux-gnueabihf/include/gnu/stubs.h
```

11. Build compiler support library

```
cd ..
cd build-gcc
make -j8 all-target-libgcc
make install-target-libgcc
```

12. Finish building Glibc

```
cd ..
cd build-glibc
make -j8
make install
```

13. Finish building GCC

```
cd ..
cd build-gcc
make -j8
make install
cd ..
```

Additional Note: If input now into your shell:

```
arm-linux-gnueabihf-g++
```

it should say something like:

```
arm-linux-gnueabihf-g++: fatal error: no input files
compilation terminated.
```

That means that arm-linux-gnueabihfg++ is in path as it should be.

1. Compiling libtensorflow-lite.a and minimal

1. Clone tensorflow:

```
git clone https://github.com/tensorflow/tensorflow.git tensorflow_s
```

2. Download dependencies tensorflow lite:

```
cd tensorflow_src && ./tensorflow/lite/tools
/make/download_dependencies.sh
```

3. Use the makefile through the build script provided by tensorflow to compile libtensorflow-lite.a and minimal:

```
./tensorflow/lite/tools/make/build_rpi_lib.sh TARGET_ARCH=armv6
TARGET_TOOLCHAIN_PREFIX=arm-linux-gnueabihf-  >errors.txt 2>&1
```

(this will also pipe errors into a file called errors.txt in the same dictionary) After the process is finished check errors.txt and check `/tensorflow/lite/tools /make/gen/` . The folder should contain a folder called rpi_armv6 containing the three folders: bin, lib and obj. libtensorflow-lite.a can be used to build python wheels while minimal can run on the pi to load a model. (its basicly a tensorflow hello world in c++)

Additional note 1:

if something breaks during this compilation process the script is not cleaning the make folder. You have to do this manually before another attempt.

```
cd tensorflow/lite/tools/make
make cleanhistory
```

Additional note 2:

We encountered a strange error of an undefined reference in

> tensorflow/lite/tools/make/downloads/flatbuffers/src/util.cpp

To get rid of this open the file in an editor of your choice and change line 199 to:

```
char abs_path[2048];
```

[This is only tested by us but just
affects the minimal and other c++ applications]

2. Building bindings for tensorflow lite

The goal here is to build tensorflow lite bindings for Python 3.7.3 on armv6. This has the advantage of not having to load the whole tensorflow libary but just the reduced tflite_runtime in python.

Its possible to build the wheels on the pi zero itself with the previously compiled binarys. Therefore the simplest way is to transfer the whole tensorflow souce folder we have been working in on to the pi zero. One way of doing this is via scp:

```
scp -r yourtensorflowsoucefolder piusername@youpiip:/yourpathonpi
```

The script which tensorflow provides searches for `linux_armv6l` while the folder we compiled to is named `rpi_armv6` . Either rename or create a symlink in `tensorflow/lite/tools/make/gen`

After this run from the tensorflow source directory:

```
./tensorflow/lite/tools/pip_package/build_pip_package.sh
```

You should now have a gen folder containing the wheels.

3. Installing Wheels or using the precompiled Wheels

In the source folder:

```
pip3 install tflite_runtime-2.5.0-cp37-cp37m-linux_armv6l.whl
```

Additional note 1: this requires maybe libraries which need to be installed manually externally via apt-get install

4. Loading a model and testing tflite_runtime

tflite_runtime is build without libatomic, which makes it nessesary to preload it:

```
LD_PRELOAD=/usr/lib/arm-linux-gnueabihf/libatomic.so.1.2.0 python3
```

after this in the python interpreter:

```
from tflite_runtime.interpreter import Interpreter
interpretera = Interpreter("./yourmodel.tflite")
interpretera.get_tensor_details()
```