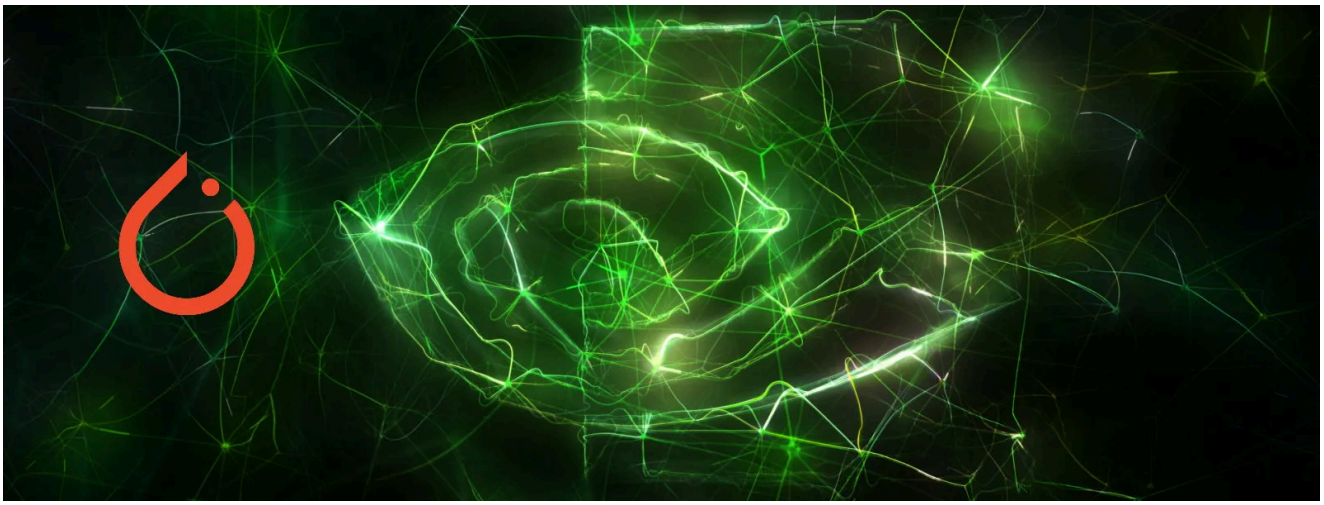


Install PyTorch on Jetson Nano.

Last updated: April 11, 2023

Pytorch 2.0 and above uses CUDA 11. The Jetson Nano has CUDA 10.2.

Due to low-level GPU incompatibility, installing CUDA 11 on your Nano is impossible.

Pytorch 2.0 can only be installed on Jetson family members using a JetPack 5.0 or higher, such as the *Jetson Nano Orion*.

Unfortunately, it does not appear that this version will also be available for the Jetson Nano soon.

Introduction.

This page will guide you through the installation of **PyTorch**, **TorchVision**, **LibTorch** and **Caffe2** on a Jetson Nano.

PyTorch is a software library specially developed for deep learning. It consumes an lot of resources of your Jetson Nano. So, don't expect miracles. It can run your models, but it can't train new models. The so-called transfer learning can cause problems due to the limited amount of available RAM.

We discuss two installations, one with a Python 3 wheel. The other method is the build from scratch. Unfortunately, there is no official pip3 wheel available for the Jetson Nano. However, we created these wheels and put them on GitHub for your convenience.

PyTorch 1.13, 1.12, 1.11.

PyTorch version 1.11 and above requires Python **3.7**, found in JetPack 5.0.

Since JetPack 4.6 has Python 3.6, you cannot install PyTorch 1.11.0 on a Jetson Nano.

It looks like Nvidia has no plans to release the new JetPack 5.0 for the Jetson Nano for now. It's only available for the Xavier series.

However, you can use the current version of Jetson Nano with Ubuntu 20.04. We supply the wheels for this version at GitHub.

PyTorch 1.10.

PyTorch 1.10 has the usual improvements and bug fixes. Please note, some operations have different behavior compared to version 1.9. Take a look at the changelog.

PyTorch 1.9.

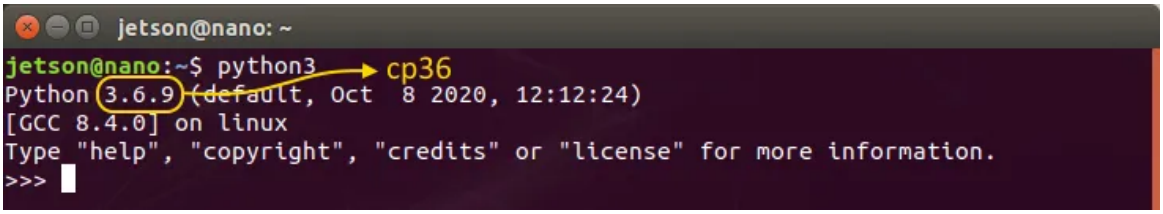
Some warnings about version 1.9.0. As seen here, quite a few changes are made to the software since the last version. Not all operations and declarations are supported anymore. It can cause backward compatibility issues when your 1.8 networks are running on this new version.

Installation by wheel.

PyTorch is build by Ninja. It takes more then 5 hours to complete the whole build. We have posted the wheels on our GitHub page. Feel free to use these. With all the tedious work already done, it takes now only a couple of minutes to install PyTorch on your Nano. For the diehards, the complete procedure is covered later in this manual.

The whole shortcut procedure is found below. The wheel was too large to store at GitHub, so Google drive is used. Please make sure you have latest pip3 and python3 version installed, otherwise, pip may come with the message ".whl is not a supported wheel on this platform".

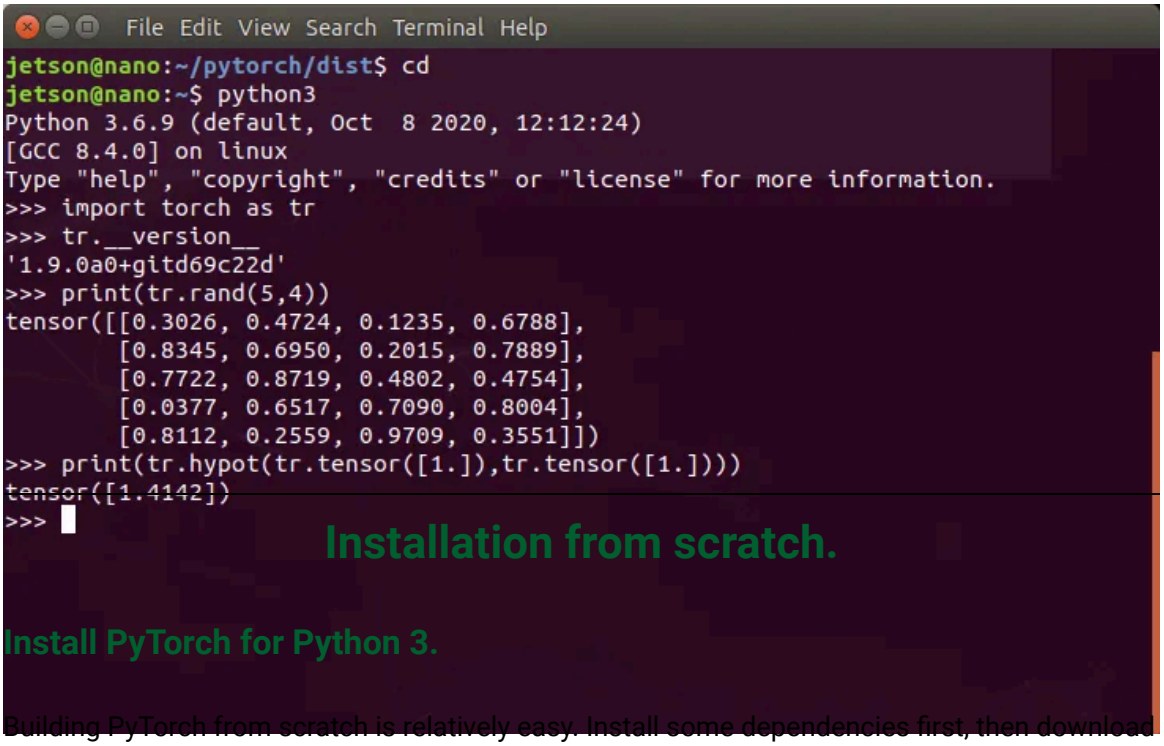
See out GitHub page for all the wheels.



PyTorch 1.13.0 | 1.12.0 | 1.11.0 | 1.10.0 | 1.9.0 | 1.8.0 | 1.7.0

```
# install the dependencies (if not already onboard)
$ sudo apt-get install python3-pip libjpeg-dev libopenblas-dev
libopenmpi-dev libomp-dev
$ sudo -H pip3 install future
$ sudo pip3 install -U --user wheel mock pillow
$ sudo -H pip3 install testresources
# above 58.3.0 you get version issues
$ sudo -H pip3 install setuptools==58.3.0
$ sudo -H pip3 install Cython
# install gdown to download from Google drive
$ sudo -H pip3 install gdown
# download the wheel
$ gdown https://drive.google.com/uc?
id=1aWuKu8eqkZwVzFFvguVuwkj0zdCir9qX
# install PyTorch 1.7.0
$ sudo -H pip3 install torch-1.7.0a0-cp36-cp36m-linux_aarch64.whl
# clean up
$ rm torch-1.7.0a0-cp36-cp36m-linux_aarch64.whl
```

After a successful installation, you can check PyTorch with the following commands.



Building PyTorch from scratch is relatively easy. Install some dependencies first, then download the zip from GitHub and finally build the software.

Note, the whole procedure takes about 8 hours on an overclocked Jetson Nano.

PyTorch 1.13.0 | 1.12.0 | 1.11.0 | 1.10.0 | 1.9.0 | 1.8.0 | 1.7.0

```
# get a fresh start
$ sudo apt-get update
$ sudo apt-get upgrade
# the dependencies
$ sudo apt-get install ninja-build git cmake
$ sudo apt-get install libjpeg-dev libopenmpi-dev libomp-dev ccache
$ sudo apt-get install libopenblas-dev libblas-dev libeigen3-dev
$ sudo pip3 install -U --user wheel mock pillow
$ sudo -H pip3 install testresources
# above 58.3.0 you get version issues
$ sudo -H pip3 install setuptools==58.3.0
$ sudo -H pip3 install scikit-build
# download PyTorch with all its libraries
$ git clone -b v1.7.0 --depth=1 --recursive
https://github.com/pytorch/pytorch.git
$ cd pytorch
# one command to install several dependencies in one go
# installs future, numpy, pyyaml, requests
```

```
# setuptools, six, typing_extensions, dataclasses
$ sudo pip3 install -r requirements.txt
```

Enlarge memory swap.

Building the full PyTorch requires more than 4 Gbytes of RAM and the 2 Gbytes of swap space delivered by zram usually found on your Jetson Nano. We have to install dphys-swapfile to get the additional space from your SD card temporarily. After the compilation, the mechanism will be removed, eliminating swapping to the SD card.

You need to increase the dphys swap beyond the regular 2048 limit. It is done by first changing the maximum boundary in `/sbin/dphys-swapfile` to 4096. Next, set the `/etc/dphys-swapfile`. The slideshow will guide you. If there is not enough swap memory, the compilation will generate obcure `CalledProcessErrors`.

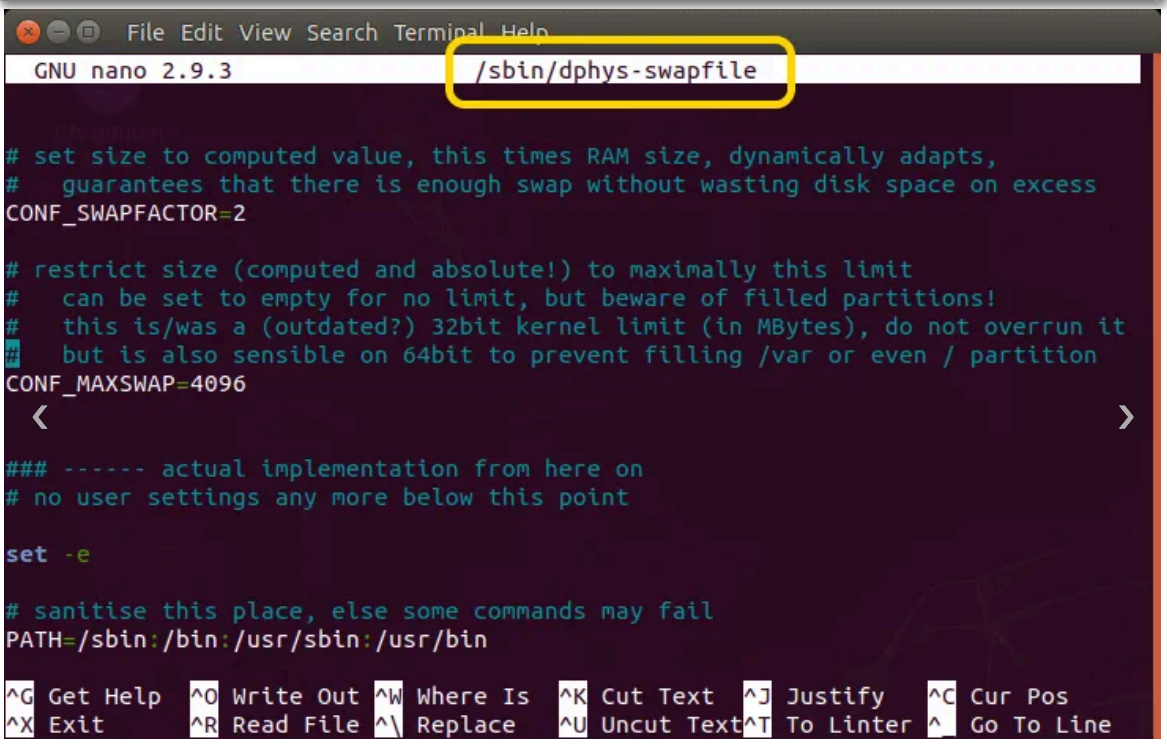
We do not recommend increasing the zram swap limits. You can't just keep compressing system memory in the hopes of getting some extra space. There are limits. It is better that you temporarily use the SD memory. Once PyTorch is installed, you can remove dphys-swapfile again.

Please follow the next commands. Note also the installation of nano, a tiny text editor.

If you don't want to swap to SD memory, you can reduce the number of working cores with the `$ export MAX_JOBS` variable. If you use two instead of four cores, the compilation will succeed without dphys-swapfile, but it will take much longer to complete. It is up to you.

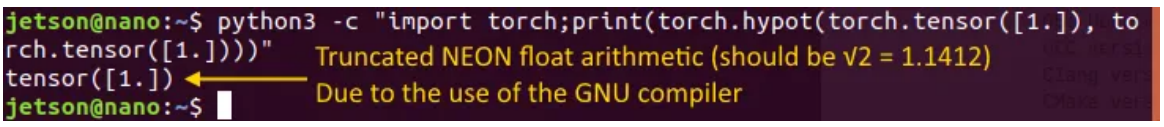
```
# a fresh start, so check for updates
$ sudo apt-get update
$ sudo apt-get upgrade
# install nano
$ sudo apt-get install nano
# install dphys-swapfile
$ sudo apt-get install dphys-swapfile
# enlarge the boundary
$ sudo nano /sbin/dphys-swapfile
```

```
# give the required memory size

$ sudo nano /etc/dphys-swapfile

# reboot afterwards

$ sudo reboot.
```

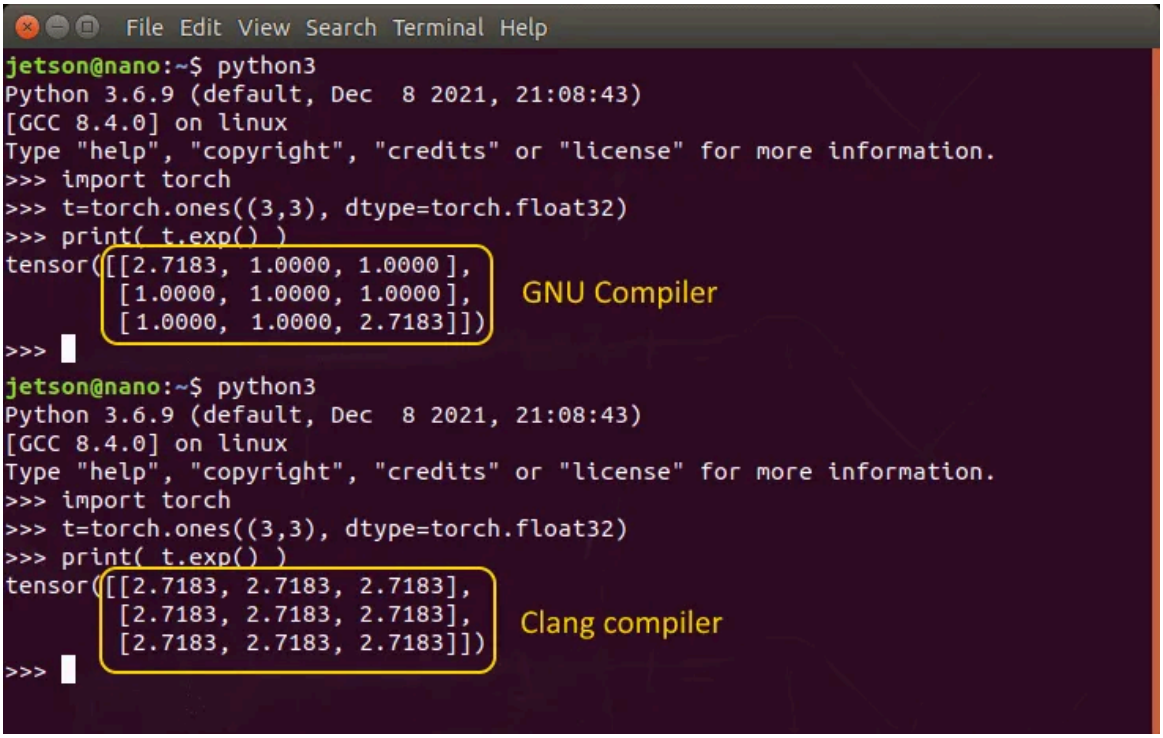


Clang compiler.

Before the build can begin, some preparations are required. First, you must have the latest clang compiler on your Jetson Nano. There is a constant stream of issues with the GNU compiler and the Jetson Nano when compiling PyTorch. Usually, it has to do with poor support of the NEON architecture of the ARM cores, causing floating points to be truncated.



Or causing floating points to be erroneous.



Oddly enough, the clang compiler doesn't seem to have a problem with the code at all, so time to use clang this time. We know there are people disliking clang. The GNU compiler used to be superior compared to clang, but those days are long gone. Today, both compilers perform almost identically.

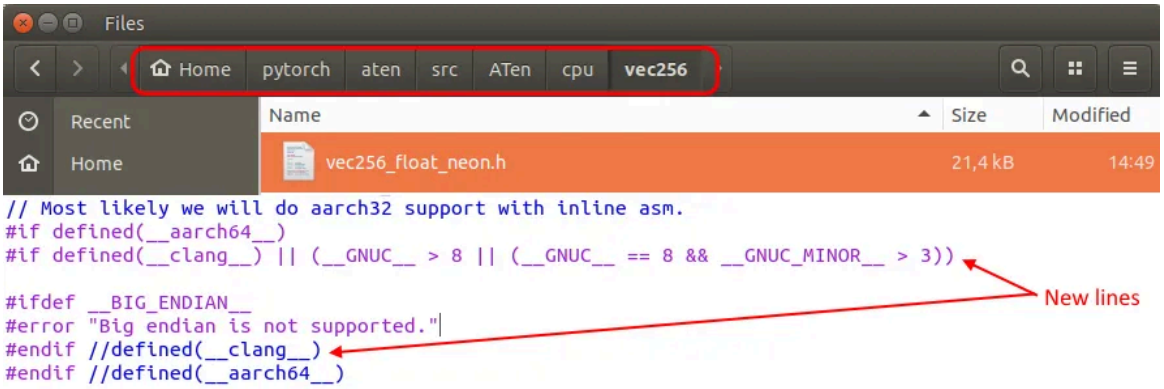
```
# install the clang compiler (version 8)

$ sudo apt-get install clang-8

# create symlinks to clang

$ sudo ln -s /usr/bin/clang-8 /usr/bin/clang

$ sudo ln -s /usr/bin/clang++-8 /usr/bin/clang++
```

Next, you have to modify the PyTorch code you just downloaded from GitHub. All alterations limits the maximum of CUDA threads available during runtime. There are four places which need our attention.

Python 1.10>: ~/pytorch/aten/src/ATen/cpu/vec/vec256/vec256_float_neon.h

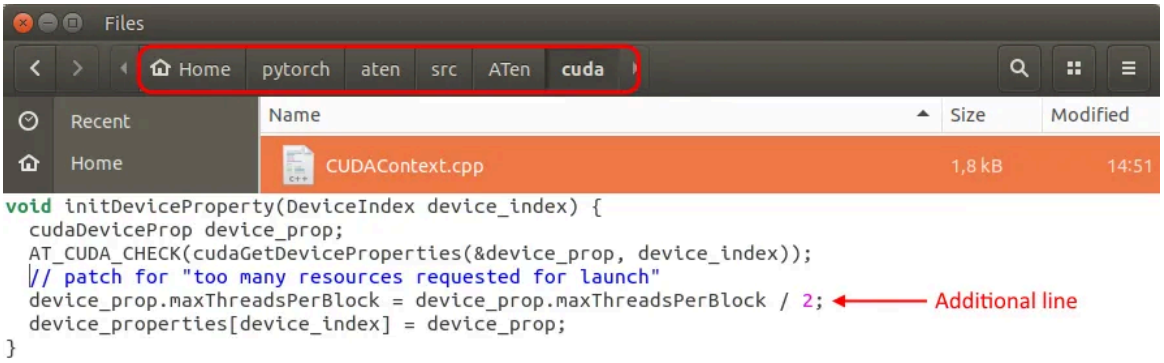
Python < 1.10 : ~/pytorch/aten/src/ATen/cpu/vec256/vec256_float_neon.h

Around line 28 add `#if defined(__clang__) ||(__GNUC__ > 8 || (__GNUC__ == 8 && __GNUC_MINOR__ > 3))` and the matching closure `#endif`.



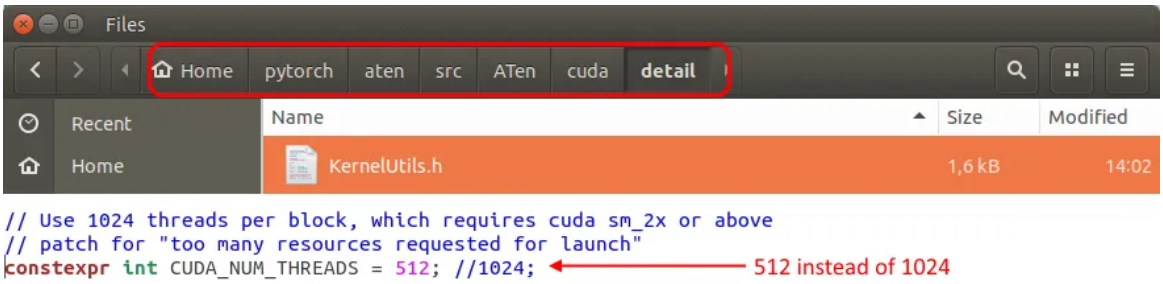
~/pytorch/aten/src/ATen/cuda/CUDAContext.cpp

Around line 24 add an extra line `device_prop.maxThreadsPerBlock = device_prop.maxThreadsPerBlock / 2;`



~/pytorch/aten/src/ATen/cuda/detail/KernelUtils.h

In line 26 change the constant from 1024 to 512.



Python 1.10> : common.h is no longer in use.

Python <1.10 : ~/pytorch/aten/src/THCUNN/common.h

In line 22 the same modification, change the `CUDA_NUM_THREADS` from 1024 to 512

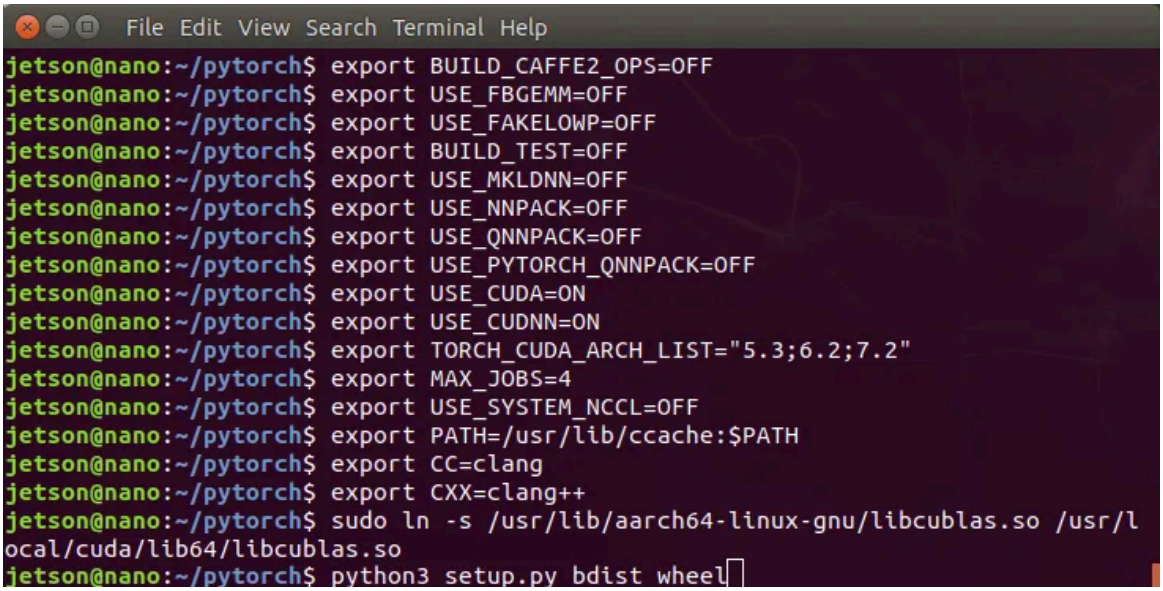
With all preparations done, we can now set the environment parameters so that the Ninja compiler gets the correct instructions on how we want PyTorch built. As you know, these instructions are only valid in the current terminal. If you start the build in another terminal, you will need to set the parameters again.

Note also the symbolic link at the end of the instructions. NVIDIA has moved the cublas library from /usr/local/cuda/lib64/ to the /usr/lib/aarch64-linux-gnu/ folder, leaving much software, like PyTorch, with broken links. A symlink is the best workaround here.

```
# set NINJA parameters
$ cd pytorch
$ export BUILD_CAFFE2_OPS=0FF
$ export USE_FBGEMM=0FF
$ export USE_FAKELOWP=0FF
$ export BUILD_TEST=0FF
```

Another noteworthy point is the arch_list. Not only is the Jetson Nano's 5.3 architectural CUDA number given, but also numbers for the Jetson Xavier. Now the wheel supports the Xavier devices also.

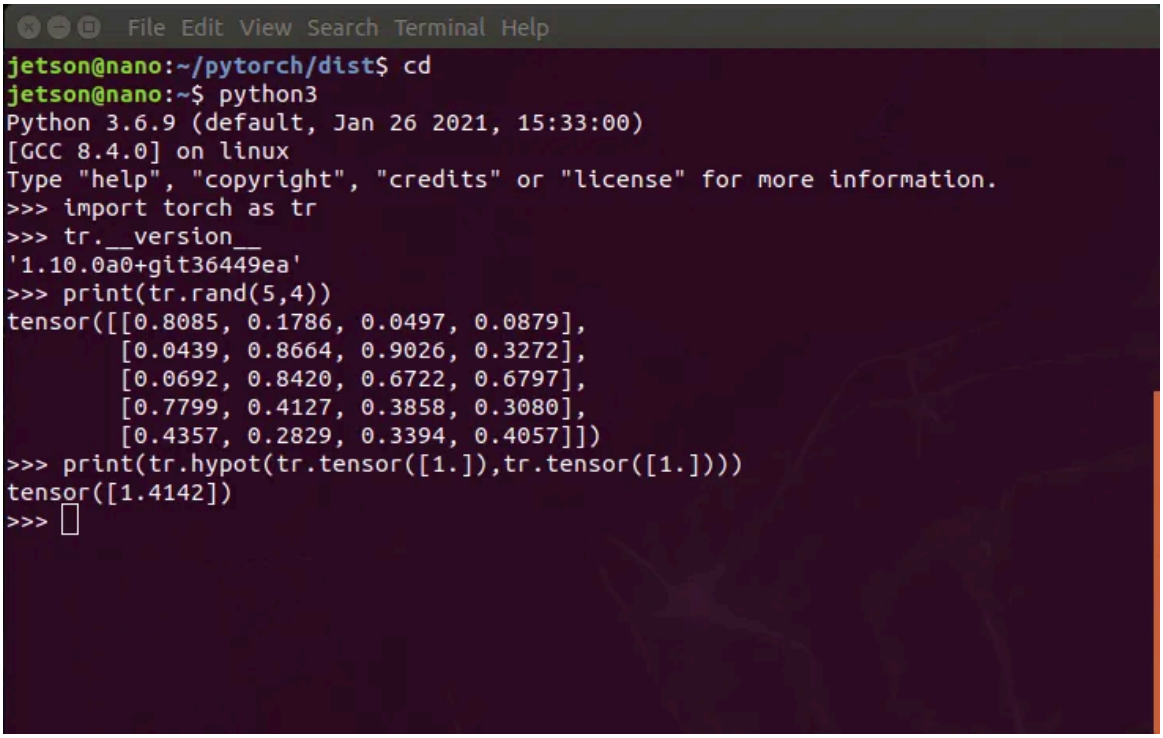
```
$ export USE_MKLDNN=OFF
$ export USE_NNPACK=OFF
$ export USE_XNNPACK=OFF
$ export USE_QNNPACK=OFF
$ export USE_PYTORCH_QNNPACK=OFF
$ export USE_CUDA=ON
$ export USE_CUDNN=ON
$ export TORCH_CUDA_ARCH_LIST="5.3;6.2;7.2"
$ export USE_NCCL=OFF
$ export USE_SYSTEM_NCCL=OFF
$ export USE_OPENCV=OFF
$ export MAX_JOBS=4
# set path to ccache
$ export PATH=/usr/lib/ccache:$PATH

# set clang compiler

$ export CC=clang
$ export CXX=clang++

# set cuda compiler
$ export CUDACXX=/usr/local/cuda/bin/nvcc
# create symlink to cublas

$ sudo ln -s /usr/lib/aarch64-linux-gnu/libcublas.so
/usr/local/cuda/lib64/libcublas.so

# clean up the previous build, if necessary
$ python3 setup.py clean
# start the build
$ python3 setup.py bdist_wheel
```



Once Ninja finished the build, you can install PyTorch on your Jetson Nano with the generated wheel. Follow the instructions below.

```
# install the wheel found in the dist folder

$ cd dist

$ ls

$ sudo -H pip3 install torch-<version>-cp36-cp36m-linux_aarch64.whl
```

After successful installation, you can check PyTorch with the commands given at the end of the previous section.



One word about OpenCV. PyTorch has the option the use OpenCV. However, it links hardcoded with the OpenCV version found during the build. As soon as you upgrade your OpenCV, PyTorch will stop working since it can't find the old OpenCV version. Given OpenCV enthusiasm to release at least two or three versions a year, it seems not wise to link PyTorch with OpenCV. Otherwise, you will be forced to recompile PyTorch or manually create a whole bunch of symbolic links to the old libraries.

```
# remove the dphys-swapfile now
$ sudo /etc/init.d/dphys-swapfile stop
$ sudo apt-get remove --purge dphys-swapfile

# just a tip to save some space

$ sudo rm -rf ~/pytorch
```

After a successful installation, many files are no longer needed. Removing them will give you about 3.6 GB of disk space.

Be sure to remove the SD memory swap software installed at the beginning of this manual.

TorchVision.

Install torchvision on Jetson Nano.

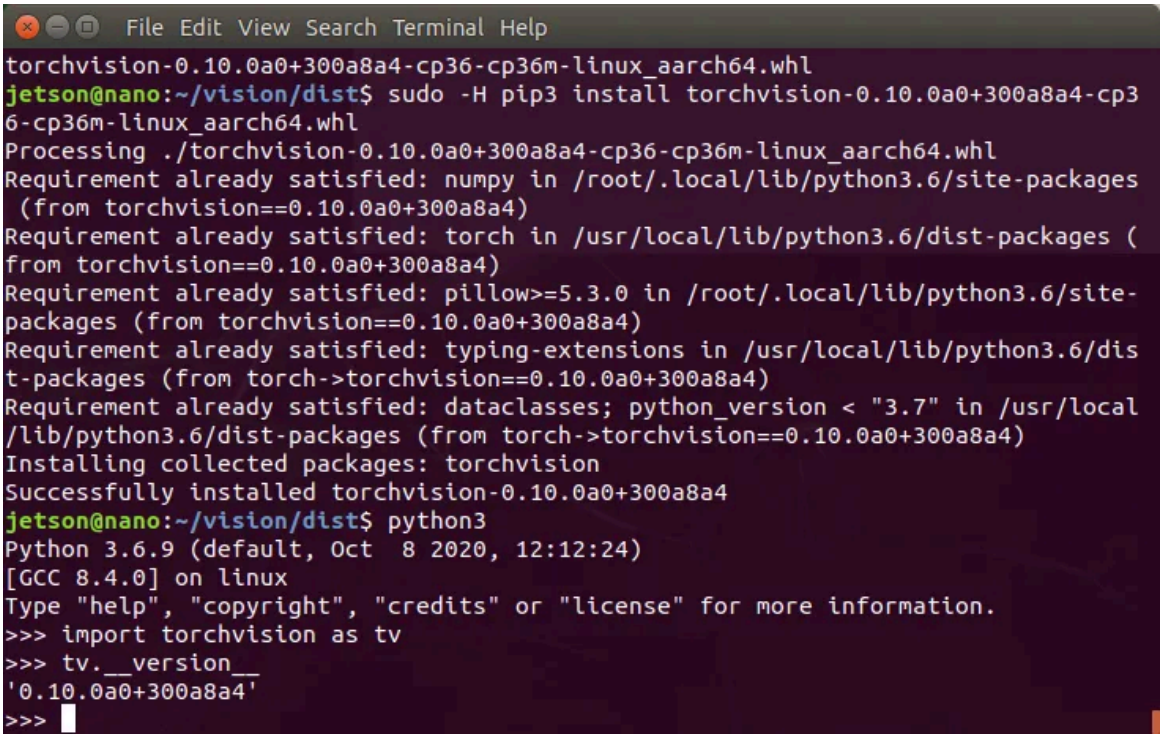
Torchvision is a collection of frequent used datasets, architectures and image algorithms. The installation is simple when you use one of our wheels found on GitHub. Torchvision assumes PyTorch is installed on your machine on the forehand.

Vision 0.14.0	0.13.0	0.12.0	0.11.0	0.10.0	0.9.0	0.8.0

Used with PyTorch **1.7.0**

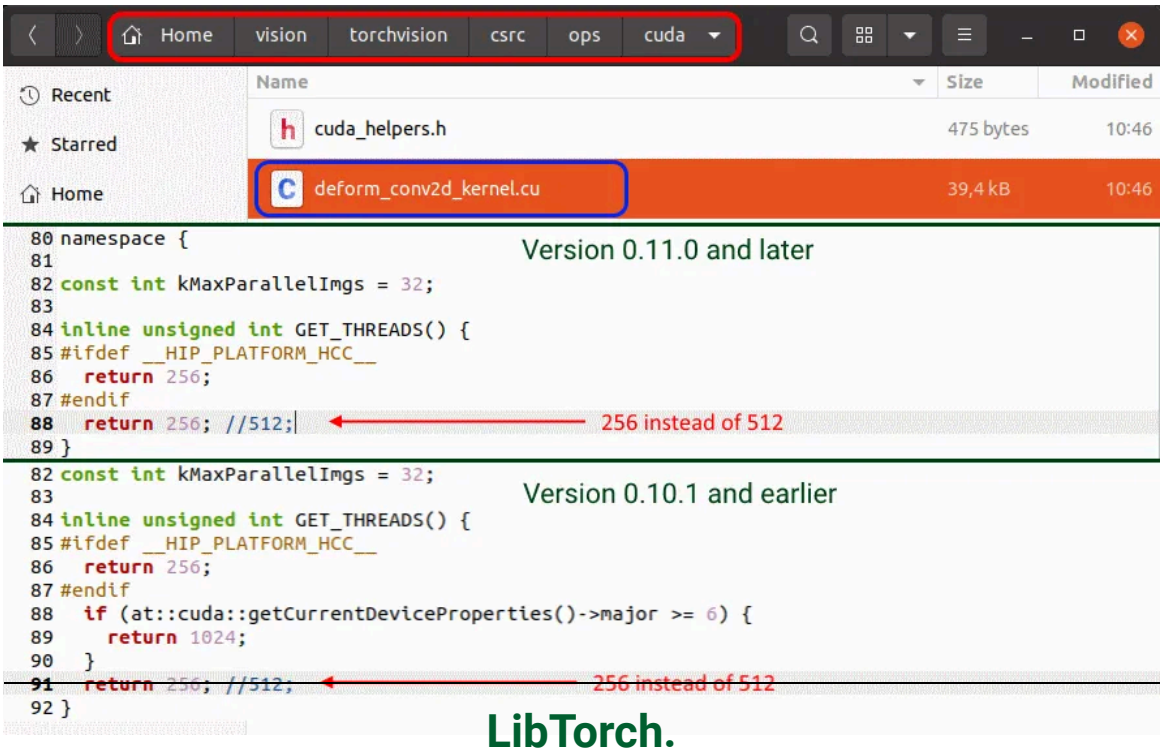
```
# the dependencies
$ sudo apt-get install libjpeg-dev zlib1g-dev libpython3-dev
$ sudo apt-get install libavcodec-dev libavformat-dev libswscale-dev
$ sudo pip3 install -U pillow
# install gdown to download from Google drive, if not done yet
$ sudo -H pip3 install gdown
# download TorchVision 0.8.0
$ gdown https://drive.google.com/uc?id=1P0xyPT-WIWglqmT195OSyazV_1LPaHDa
# install TorchVision 0.8.0
$ sudo -H pip3 install torchvision-0.8.0a0+291f7e2-cp36-cp36m-linux_aarch64.whl
# clean up
$ rm torchvision-0.8.0a0+291f7e2-cp36-cp36m-linux_aarch64.whl
```

After installation you may want to check torchvision by verifying the release version.



You can also build torchvision from scratch. In that case, you have to download the version of your choice from the official GitHub page, modify the version number in version.txt and issue the command `$ python3 setup.py bdist_wheel`.

Brett Ryland, from Bergen Robotics AS, kindly emailed us, pointing out that one constant in the CUDA code needs to be changed to work with deformable convolutions. Open the deform_conv2d_kernel.cu file with an editor and lower the number of threads, as shown below.



LibTorch.

Install LibTorch on Jetson Nano.

The native language of PyTorch is Python. For a good reason.

AI scientists want to mold their deep learning models and analyses the outcomes without the hassle of dedicated software programming.

Python is well suited for this job. Most people can understand and modify a Python program in a few weeks, while it takes years to grasp the subtleties of the (low-level) C++ language. If you are new to deep learning and PyTorch, we strongly recommend using Python.

There are more things to know before starting your C++ adventure.

- The precompiled LibTorch is only suitable for an x86_64 machine. There is no aarch64 version, we had to build it from scratch.

- The C++ documentation is of low quality. You have a brief explanation of the function calls, but a good guide on installing LibTorch and what to do if something goes wrong is missing. (By the way, most frameworks have the same lack of documentation.)

- Compilation times are significant on a bare Jetson Nano. A simple example.cpp, shown later, takes about a minute to build. It is better to use cross-compilation techniques if you are serious about programming in C++. Waiting more than a minute to correct a simple typo is frustrating.

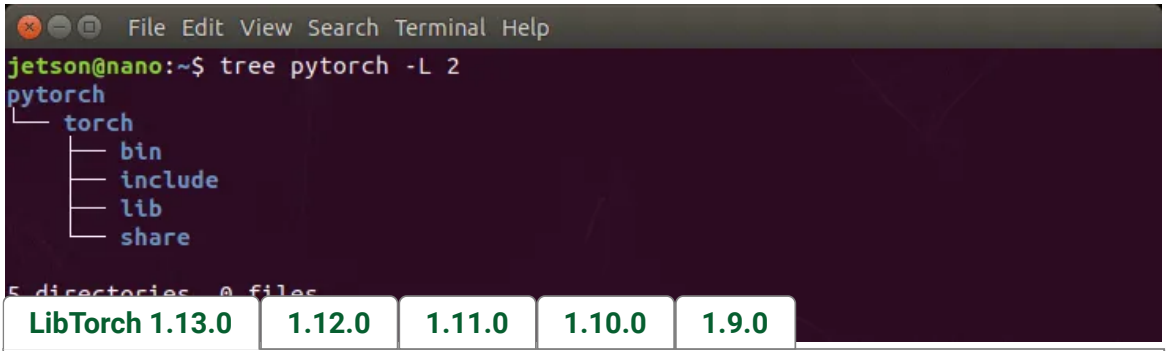
- The core of PyTorch is built with an old GCC compiler, not using the 2011 C++ naming convention for strings. To use the static build LibTorch, you must use the macro _GLIBCXX_USE_CXX11_ABI, which may conflict with other parts of your C++ program. If you want to use LibTorch, better use the dynamic build version. It can work smoothly with other C++ packages, such as ROS.

- After three days of toil, we still couldn't get the static build LibTorch to work on the Jetson Nano. By scrolling through the forums, you will read the same questions and the same (poor) solutions about LibTorch static links over and over.

- There are problems with the GCC optimization of intrinsics of the NEON registers of the ARM core, as shown above. It forces us to use the clang compiler. Therefore, when using the LibTorch API, you may be forced to use the clang compiler as well.

Now, let's start building the LibTorch C++ API.

There are two possible ways to install LibTorch on your Jetson Nano. The first method is to download the tar.xz file from our GitHub and extract it. All necessary libraries and headers are installed, as seen in the screenshot below.



LibTorch 1.13.0 | 1.12.0 | 1.11.0 | 1.10.0 | 1.9.0

Only for a Jetson Nano with **Ubuntu 20.04**

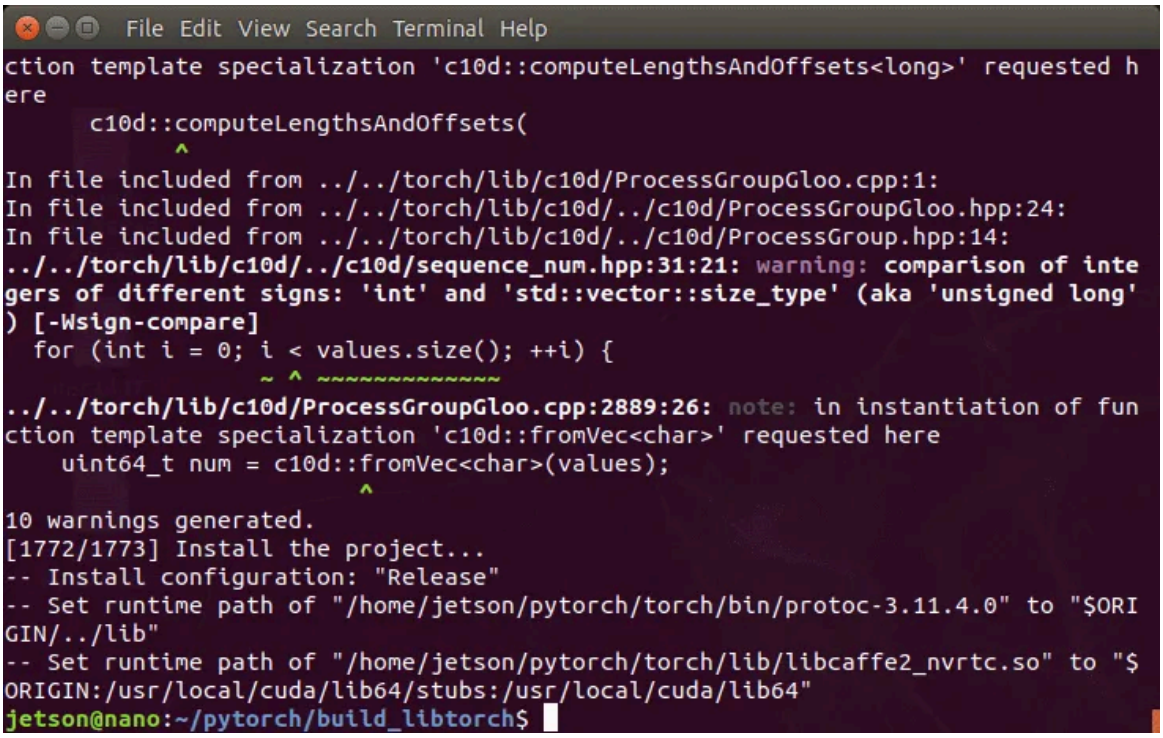
```
# install gdown to download from Google drive, if not done yet
$ sudo -H pip3 install gdown
# download LibTorch 1.13.0
$ gdown https://drive.google.com/uc?
id=1k8nDUFUI_5_07MKkZTJzK4o1TEtks9mQ
# unpack the LibTorch 1.13.0 tar ball
$ sudo tar -xf libtorch-1.13.0-Jetson-aarch64-GPU.tar.gz
# clean up
$ rm libtorch-1.13.0-Jetson-aarch64-GPU.tar.gz
```

The other way is to compile the LibTorch C++ API from scratch. The whole procedure is almost identical to the original Python installation. Follow the instructions if you want to compile and install the libraries from scratch. If you want static libraries (libtorch.a), set the environment flag `BUILD_SHARED_LIBS=OFF`. As mentioned, we couldn't get the static libraries to work on the Nano. Better to use the dynamic libraries (libtorch.so).

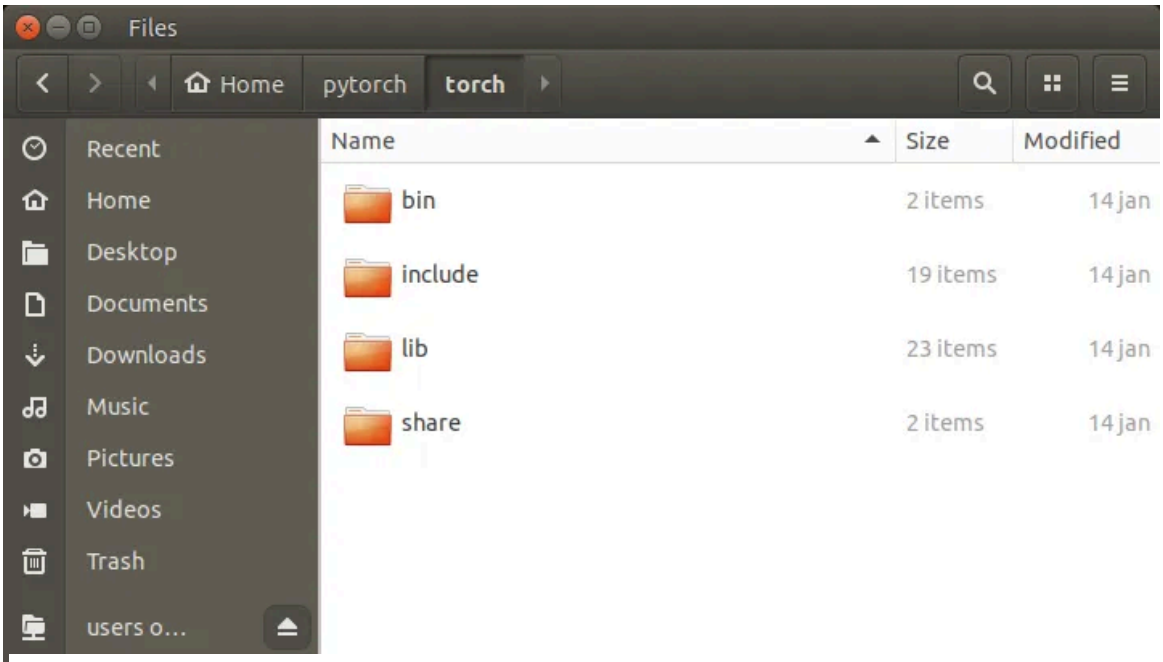
```
# First, download and install the dependencies and
# your PyTorch version of your choice as specified above.
# Follow all steps up until the environment variables.
# Don't forget to modify the five files also.


$ cd ~/pytorch
$ mkdir build_libtorch
$ cd build_libtorch
# now set the temporary environment variables for LibTorch

# remember, don't close the window as it will delete these variables
```

```
$ export BUILD_PYTHON=OFF
$ export BUILD_CAFFE2_OPS=OFF

$ export USE_FBGEMM=OFF

$ export USE_FAKELOWP=OFF

$ export BUILD_TEST=OFF

$ export USE_MKLDNN=OFF

$ export USE_NNPACK=OFF

$ export USE_XNNPACK=OFF

$ export USE_QNNPACK=OFF

$ export USE_PYTORCH_QNNPACK=OFF
$ export USE_CUDA=ON
$ export USE_CUDNN=ON
$ export TORCH_CUDA_ARCH_LIST="5.3;6.2;7.2"
$ export MAX_JOBS=4

$ export USE_NCCL=OFF

$ export USE_OPENCV=OFF
$ export USE_SYSTEM_NCCL=OFF

$ export BUILD_SHARED_LIBS=ON
$ PATH=/usr/lib/ccache:$PATH
# set the compilers
$ export CC=clang
$ export CXX=clang++

$ export CUDACXX=/usr/local/cuda/bin/nvcc

# create symlink to cublas (if not done yet)

$ sudo ln -s /usr/lib/aarch64-linux-gnu/libcublas.so
/usr/local/cuda/lib64/libcublas.so

# clean up the previous build, if necessary
```

```
$ python3 setup.py clean

# start the build

$ python3 ./tools/build_libtorch.py
```



When the build is complete, you may want to strip the ~/pytorch directory. It will save a lot of disk space. The only folder you need is the ~pytorch/torch folder. In this directory, you can delete everything except the bin, include, lib and the share folder. Don't forget the many hidden files. You end up with the same structure as shown in the tar.gz installation above.




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

using namespace std;

int main()
{
    torch::Tensor tensor = torch::rand({2, 3});
    cout << tensor << endl;
    cout << torch::hypot(torch::tensor(1.),torch::tensor(1.))<< endl;

    auto t = torch::ones({3, 3}, torch::dtype(torch::kFloat32));
    cout << "t:\n" << t << endl;
    cout << "t.exp():\n" << t.exp() << endl;

    return 0;
}
```

We're using the CMake file on the same PyTorch page. We only stripped the Windows MSVC branch as it is not needed in a Linux environment.

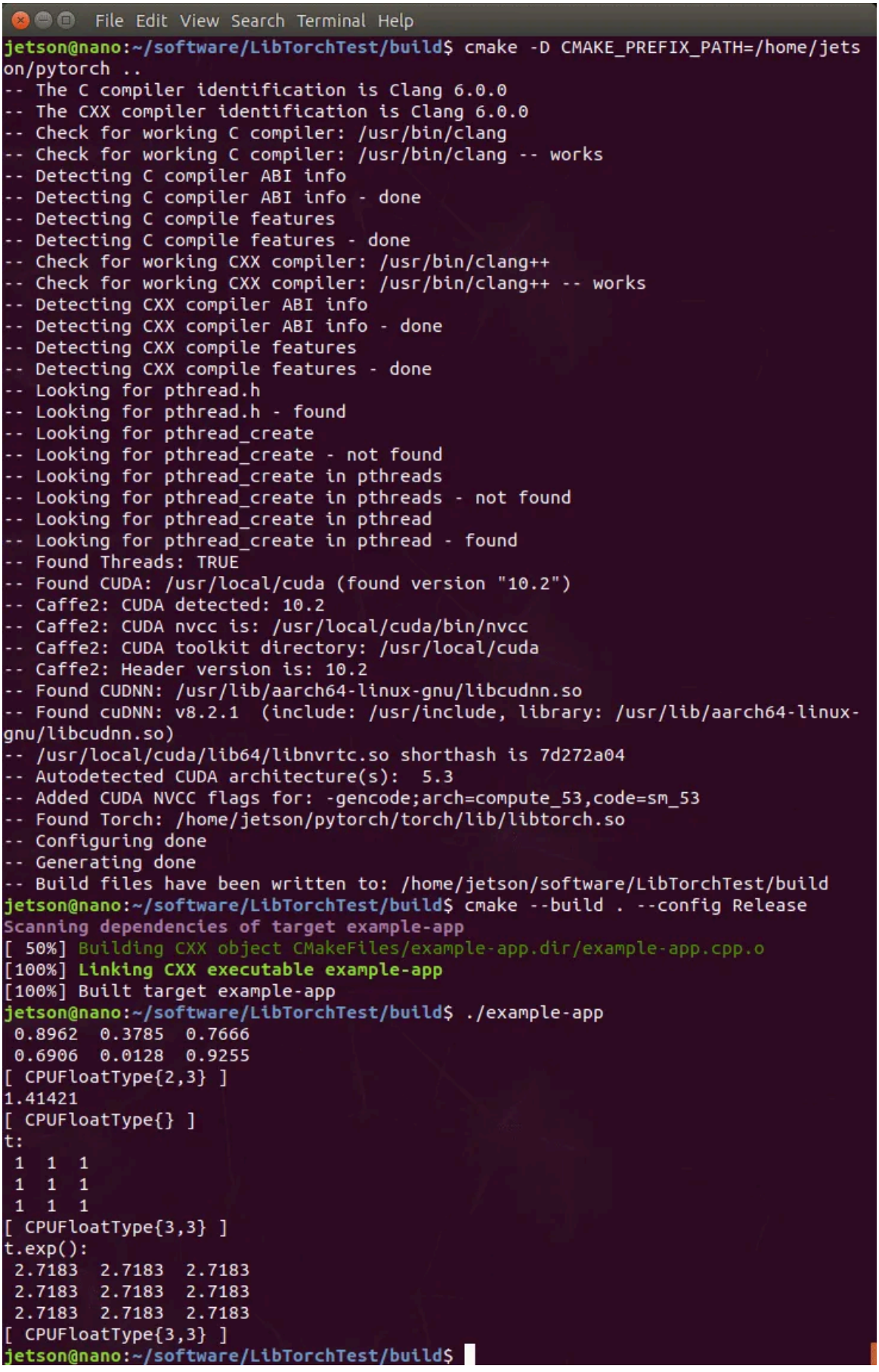
Save the file as CMakeLists.txt in the same folder as where you have placed your example-app.cpp.

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

```
set(CMAKE_C_COMPILER clang)
set(CMAKE_CXX_COMPILER clang++)

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)
```

Then create a folder called build and in this directory create the application with the following commands.

```
# make the build folder

$ mkdir build

$ cd build

$ cmake -D CMAKE_PREFIX_PATH=/home/pi/pytorch ..

$ cmake --build . --config Release
```


File Edit View Search Terminal Help
jetson@nano:~/software/LibTorchTest/build$ cmake -D CMAKE_PREFIX_PATH=/home/jetson/pytorch ..
-- The C compiler identification is Clang 6.0.0
-- The CXX compiler identification is Clang 6.0.0
-- Check for working C compiler: /usr/bin/clang
-- Check for working C compiler: /usr/bin/clang -- 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/clang++
-- Check for working CXX compiler: /usr/bin/clang++ -- 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
-- Found CUDA: /usr/local/cuda (found version "10.2")
-- Caffe2: CUDA detected: 10.2
-- Caffe2: CUDA nvcc is: /usr/local/cuda/bin/nvcc
-- Caffe2: CUDA toolkit directory: /usr/local/cuda
-- Caffe2: Header version is: 10.2
-- Found CUDNN: /usr/lib/aarch64-linux-gnu/libcudnn.so
-- Found cuDNN: v8.2.1 (include: /usr/include, library: /usr/lib/aarch64-linux-gnu/libcudnn.so)
-- /usr/local/cuda/lib64/libnvrtc.so shorthash is 7d272a04
-- Autodetected CUDA architecture(s): 5.3
-- Added CUDA NVCC flags for: -gencode;arch=compute_53,code=sm_53
-- Found Torch: /home/jetson/pytorch/torch/lib/libtorch.so
-- Configuring done
-- Generating done
-- Build files have been written to: /home/jetson/software/LibTorchTest/build
jetson@nano:~/software/LibTorchTest/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
jetson@nano:~/software/LibTorchTest/build$ ./example-app
0.8962 0.3785 0.7666
0.6906 0.0128 0.9255
[CPUFloatType{2,3}]
1.41421
[CPUFloatType{}]
t:
1 1 1
1 1 1
1 1 1
[CPUFloatType{3,3}]
t.exp():
2.7183 2.7183 2.7183
2.7183 2.7183 2.7183
2.7183 2.7183 2.7183
[CPUFloatType{3,3}]
jetson@nano:~/software/LibTorchTest/build$

More information about the C++ API Library can be found on the PyTorch site. You can also find guides on how to transfer your TorchScript to C++ here.

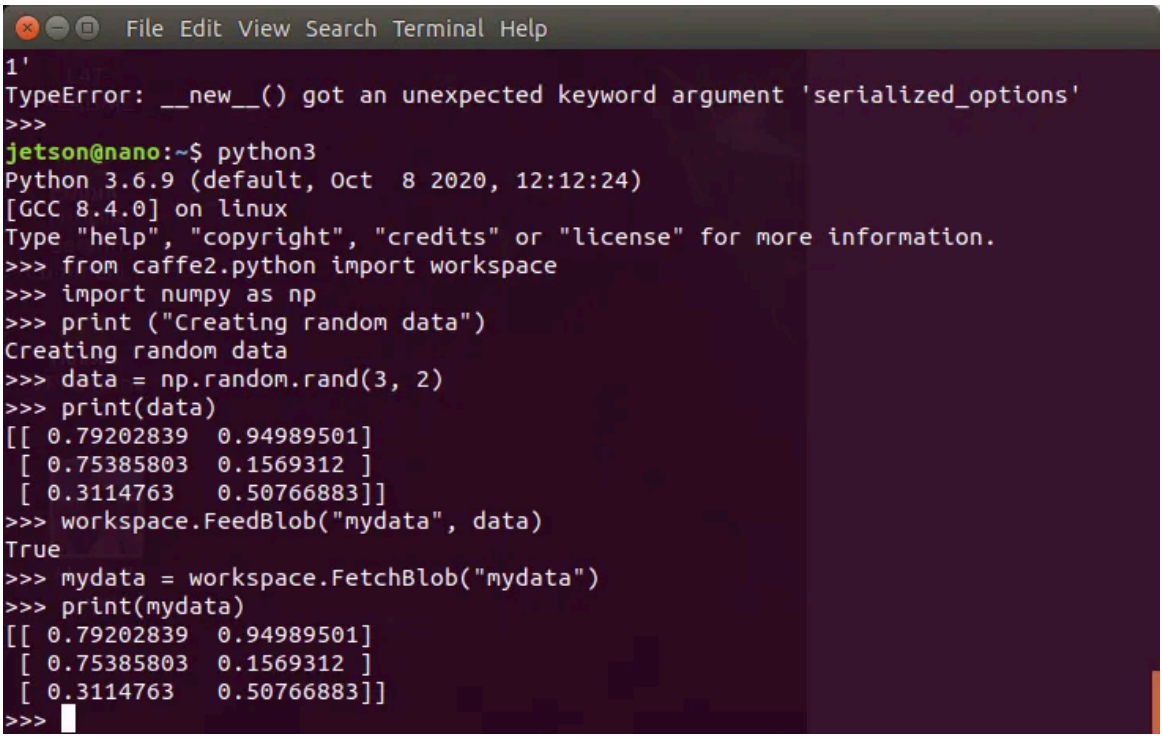
Caffe2.

Install Caffe2 on Jetson Nano.

PyTorch comes with Caffe2 on board. In other words, if you have PyTorch installed, you have also installed Caffe2 with CUDA support on your Jetson Nano. Together with two conversion tools. Before using Caffe2, most of the time protobuf needs to be updated. Let's do it right away now.

```
# update protobuf (3.15.5)

$ sudo -H pip3 install -U protobuf
```

You can check the installation of Caffe2 with a few Python instructions.


File Edit View Search Terminal Help
1'
TypeError: __new__() got an unexpected keyword argument 'serialized_options'
>>>
jetson@nano:~$ python3
Python 3.6.9 (default, Oct 8 2020, 12:12:24)
[GCC 8.4.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> from caffe2.python import workspace
>>> import numpy as np
>>> print ("Creating random data")
Creating random data
>>> data = np.random.rand(3, 2)
>>> print(data)
[[0.79202839 0.94989501]
[0.75385803 0.1569312]
[0.3114763 0.50766883]]
>>> workspace.FeedBlob("mydata", data)
True
>>> mydata = workspace.FetchBlob("mydata")
>>> print(mydata)
[[0.79202839 0.94989501]
[0.75385803 0.1569312]
[0.3114763 0.50766883]]
>>>