



How to install TensorFlow with GPU support on Ubuntu 18.04 LTS + CUDA 10



Christian Janze [Follow](#)

Dec 19, 2018 · 8 min read ★

The purpose of this guide is to demonstrate how to install [TensorFlow](#) on [Ubuntu 18.04 LTS](#) using [CUDA-enabled NVIDIA GPU's](#). No prior experience with Ubuntu or TensorFlow is required.

Please note: In the following, all commands starting with a `$` sign (do not copy the dollar sign when copying commands) are expected to be executed within a command line interface tool such as Terminal (which can be opened on Ubuntu by pressing Ctrl+Alt+t).

Update (December 27, 2018): I added a troubleshooting section at the very bottom of the article. Please comment if you encounter other issues so I can add them to the troubleshooting section.

Step 0: Prerequisites

- **Ubuntu 18.04 LTS.** You can verify your Ubuntu version with the command:

```
$ lsb_release -a
```

- **CUDA-enabled GPU from NVIDIA.** While nearly all newer NVIDIA GPU's are CUDA-enabled, please verify this for your specific GPU model [here](#).

- **Installed GNU Compiler Collection (GCC).** You can verify that you have GCC installed by issuing the command:

```
$ dpkg -s gcc
```

In case you get the message “dpkg-query: package ‘gcc’ is not installed and no information is available ...”, please install it with the command:

```
$ sudo apt update && sudo apt install gcc
```

- **Installed build-essential package.** You can verify that you have the build-essential package installed by issuing the command:

```
$ dpkg -s build-essential
```

In case you get the message “dpkg-query: package ‘build-essential’ is not installed ...”, please install it with the command:

```
$ sudo apt update && sudo apt install build-essential
```

- **Installed OpenGL Utility Toolkit (FreeGlut) and X11 Input extension library.** To install these packages, please enter the following command:

```
$ sudo apt update && sudo apt install freeglut3
```

```
freeglut3-dev libxi-dev libxmu-dev
```

- **Installed pip3 and virtualenv.** To install these packages, run the following commands subsequently:

```
$ sudo apt update && sudo apt install python3-dev
```

```
python3-pip
```

```
$ sudo pip3 install -U virtualenv
```

Step 1: Installing NVIDIA's Linux Display Driver via Runfile

Locate and Download NVIDIA's Linux Display Driver: Navigate to [NVIDIA's Driver Download](#) page, and enter the details of your specific GPU. Select Linux 64-bit as your operating system and click on the green search button (see Exhibit 1). On the following two pages, please click on the green download buttons and save the file to your ~/Downloads folder.

NVIDIA Driver Downloads

Option 1: Manually find drivers for my NVIDIA products. [Help](#)

Product Type:

Product Series:

Product:

Operating System:

Language:

Exhibit 1: NVIDIA Driver Download Configuration (Example)

Install NVIDIA's Display Driver: Once the file NVIDIA-Linux-x86_64-[YOURVERSION].run file is downloaded, please navigate to the ~/Downloads folder within the Terminal (`$ cd ~/Downloads`) and run the following shell script :

```
$ sudo sh ./NVIDIA-Linux-X86_64-[YOURVERSION].run
```

As noted by NVIDIA, at the end of the installation, you can either directly allow the modification of the X configuration file, do it manually or run `$ nvidia-xconfig`. My recommendation is to allow the X configuration modification by the installer.

Restart Machine and Verify: Once the Display Driver is installed, restart your system and verify the installation by executing the NVIDIA System Management Interface:

```
$ nvidia-smi
```

... which produces an overview of different metrics as shown in Exhibit 2.

```
Wed Dec 19 08:59:31 2018
+-----+
| NVIDIA-SMI 410.78      | Driver Version: 410.78      | CUDA Version: 10.0      |
+-----+-----+
| GPU   Name           Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf    Pwr:Usage/Cap|     Memory-Usage | GPU-Util  Compute M. |
+-----+-----+
|  0 GeForce RTX 2070    Off      | 00000000:07:00.0 On      |         N/A         |
|  0%    41C    P0     60W / 175W | 284MiB / 7949MiB |      5%      Default |
+-----+-----+
```

Exhibit 2: Example Output of nvidia-smi Tool

Step 2: Installing CUDA Toolkit 10 via Runfile

“The NVIDIA CUDA Toolkit provides command-line and graphical tools for building, debugging and optimizing the performance of applications accelerated by NVIDIA GPUs, runtime and math libraries, and documentation including programming guides, user manuals, and API references” (Source: NVIDIA CUDA Toolkit 10 installer, 2018).

Locate and Download NVIDIA’s CUDA Toolkit 10: Head over to [NVIDIA’s CUDA Toolkit Download](#) page. The suitable selections should already be made (see Exhibit 3). Then, please click on the green download button and download the cuda_10.[YOURVERSION]_linux.run file to your ~/Downloads folder.

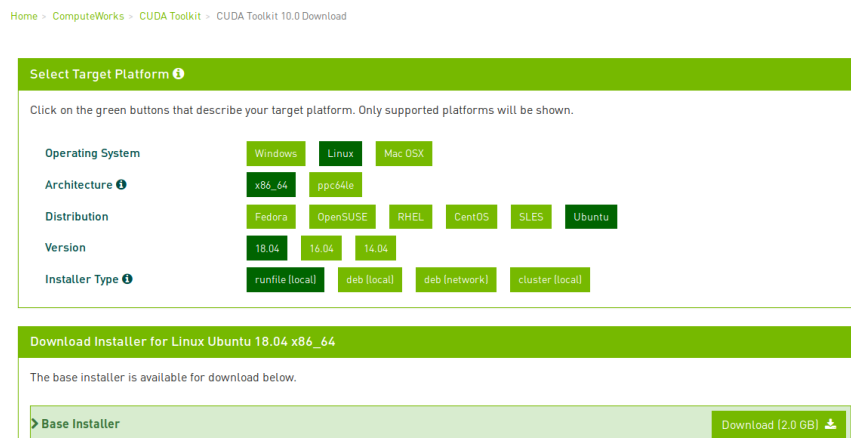


Exhibit 3: CUDA Toolkit 10 Locator

Install NVIDIA’s CUDA Toolkit 10 via Runfile: Once the file NVIDIA-Linux-x86_64-[YOURVERSION].run file is downloaded, navigate to the ~/Downloads folder and run the command:

```
$ sudo sh cuda_10.[YOURVERSION]_linux.run
```

To skip the license agreement, press `Ctrl+c`

IMPORTANT: During the installation, type “no” when asked if you want to “Install NVIDIA Accelerated Graphics Driver for Linux-x86_64 [YOURVERSION]?”. This is because, as of December 19th, 2018, the NVIDIA CUDA Toolkit 10 installer ships with an older NVIDIA Display Driver (version 410.48) than is offered on the NVIDIA Display Driver

download page from step 1 (version 410.78). Therefore, the NVIDIA CUDA Toolkit 10 installer finishes with a warning message *****WARNING: Incomplete installation! This installation did not install the CUDA Driver. ...**”.

Setup the environment variables: To do so, please open the `.bashrc` file in the editor nano by entering:

```
$ sudo nano ~/.bashrc
```

Scroll down and add the following two paths to the `.bashrc` file:

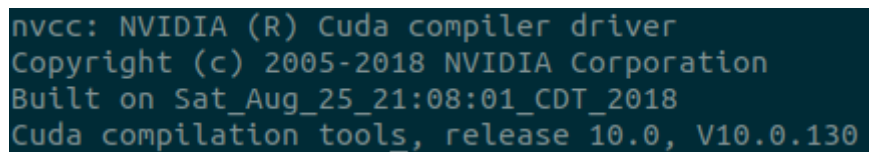
```
export PATH=/usr/local/cuda-10.0/bin${PATH:+:${PATH}}
export LD_LIBRARY_PATH=/usr/local/cuda-
10.0/lib64${LD_LIBRARY_PATH:+:${LD_LIBRARY_PATH}}
```

To save the changes, please press `Ctrl + o`, press `enter` to accept the changes and `Ctrl + x` to close nano. Then, to reload the `.bashrc` file with the changes made, please enter the following command to the Terminal:

```
$ source ~/.bashrc
```

Verify the installation of NVIDIA's CUDA Toolkit 10 compiler driver: To do so, please enter the following command (the output should look similar to Exhibit 4):

```
$ nvcc -V
```



```
nvcc: NVIDIA (R) Cuda compiler driver
Copyright (c) 2005-2018 NVIDIA Corporation
Built on Sat_Aug_25_21:08:01_CDT_2018
Cuda compilation tools, release 10.0, V10.0.130
```

Exhibit 4: Example of CUDA compiler driver install

Test the installation of NVIDIA's CUDA Toolkit 10: For this, enter the following commands to compile the CUDA 10 samples, which will take

a while:

```
$ cd ~/NVIDIA_CUDA-10.0_Samples
$ make
```

Once you see the message “Finished building CUDA samples”, enter the following commands to test CUDA GPU jobs with the deviceQuery program:

```
$ cd ~/NVIDIA_CUDA-
10.0_Samples/bin/x86_64/linux/release
$ ./deviceQuery
```

The result should look similar to Exhibit 5. Note the Result = PASS at the very bottom. If you see this message, congratulations, you have successfully installed CUDA Toolkit 10!

```
./deviceQuery Starting...

CUDA Device Query (Runtime API) version (CUDART static linking)
Detected 1 CUDA Capable device(s)

Device 0: "GeForce RTX 2070"
  CUDA Driver Version / Runtime Version      10.0 / 10.0
  CUDA Capability Major/Minor version number: 7.5
  Total amount of global memory:             7949 MBytes (8335327232 bytes)
  (36) Multiprocessors, ( 64) CUDA Cores/MP: 2304 CUDA Cores
  GPU Max Clock rate:                       1620 Mhz (1.62 GHz)
  Memory Clock rate:                        7001 Mhz
  Memory Bus Width:                         256-bit
  L2 Cache Size:                            4194304 bytes
  Maximum Texture Dimension Size (x,y,z)    1D=(131072), 2D=(131072, 65536), 3D=(16384, 16384, 16384)
  Maximum Layered 1D Texture Size, (num) layers 1D=(32768), 2048 layers
  Maximum Layered 2D Texture Size, (num) layers 2D=(32768, 32768), 2048 layers
  Total amount of constant memory:           65536 bytes
  Total amount of shared memory per block:   49152 bytes
  Total number of registers available per block: 65536
  Warp size:                                32
  Maximum number of threads per multiprocessor: 1024
  Maximum number of threads per block:      1024
  Max dimension size of a thread block (x,y,z): (1024, 1024, 64)
  Max dimension size of a grid size    (x,y,z): (2147483647, 65535, 65535)
  Maximum memory pitch:                     2147483647 bytes
  Texture alignment:                        512 bytes
  Concurrent copy and kernel execution:     Yes with 3 copy engine(s)
  Run time limit on kernels:                 Yes
  Integrated GPU sharing Host Memory:        No
  Support host page-locked memory mapping:  Yes
  Alignment requirement for Surfaces:        Yes
  Device has ECC support:                    Disabled
  Device supports Unified Addressing (UVA):  Yes
  Device supports Compute Preemption:        Yes
  Supports Cooperative Kernel Launch:        Yes
  Supports MultiDevice Co-op Kernel Launch: Yes
  Device PCI Domain ID / Bus ID / location ID: 0 / 7 / 0
  Compute Mode:
    < Default (multiple host threads can use ::cudaSetDevice() with device simultaneously) >

deviceQuery, CUDA Driver = CUDART, CUDA Driver Version = 10.0, CUDA Runtime Version = 10.0, NumDevs = 1
Result = PASS
```

Exhibit 5: Example Output of the deviceQuery Application

Step 3: Installing NVIDIA cuDNN

“The NVIDIA CUDA Deep Neural Network library (cuDNN) is a GPU-accelerated library of primitives for deep neural networks.” (Source: [NVIDIA 2018](#))

Locate and Download NVIDIA’s cuDNN 7.4: Please head over to the [NVIDIA cuDNN website](#) and click on the green Download cuDNN button. Unfortunately, you have to register for the NVIDIA Developer Program. Once you are registered, go to the [NVIDIA cuDNN website](#) and fill in the survey to reach the download page (see Exhibit 6).

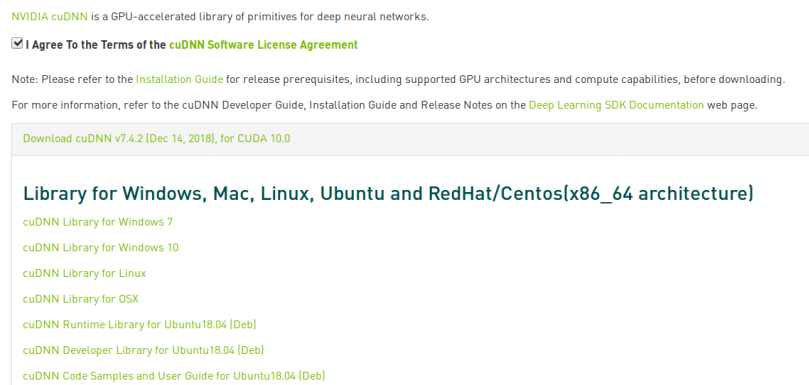


Exhibit 6: cuDNN Download Page

Here, please click on Download CuDNN v7.4-[CURRENTVERSION], for CUDA 10.0, and then download the following three .deb files for Ubuntu 18.04 to your ~/Downloads folder:

```
- cuDNN Runtime Library for Ubuntu18.04 (Deb)
- cuDNN Developer Library for Ubuntu18.04 (Deb)
- cuDNN Code Samples and User Guide for Ubuntu18.04 (Deb)
```

Install NVIDIA’s cuDNN 7.4: To install the three cuDNN packages, please execute the following commands one after another:

```
$ cd ~/Downloads
$ sudo dpkg -i libcudnn7_7.4.2.24-1+cuda10.0_amd64.deb
$ sudo dpkg -i libcudnn7-dev_7.4.2.24-1+cuda10.0_amd64.deb
$ sudo dpkg -i libcudnn7-doc_7.4.2.24-1+cuda10.0_amd64.deb
```

Verify the cuDNN installation: Please compile the MNIST example by executing the following commands as suggested by NVIDIA:

```
$ cp -r /usr/src/cudnn_samples_v7/ $HOME
$ cd $HOME/cudnn_samples_v7/mnistCUDNN
$ make clean && make
$ ./mnistCUDNN
```

If cuDNN is properly installed and running on your system, you will see a message “Test passed” as shown in Exhibit 7.

```
Performing forward propagation ...
Testing cudnnGetConvolutionForwardAlgorithm ...
Fastest algorithm is Algo 0
Testing cudnnFindConvolutionForwardAlgorithm ...
**** CUDNN_STATUS_SUCCESS for Algo 0: 0.012288 time requiring 0 memory
**** CUDNN_STATUS_SUCCESS for Algo 2: 0.028800 time requiring 57600 memory
**** CUDNN_STATUS_SUCCESS for Algo 1: 0.038912 time requiring 3464 memory
**** CUDNN_STATUS_SUCCESS for Algo 4: 0.061440 time requiring 207360 memory
**** CUDNN_STATUS_SUCCESS for Algo 7: 0.070848 time requiring 2057744 memory
Resulting weights from Softmax:
0.0000000 0.9999399 0.0000000 0.0000000 0.0000561 0.0000000 0.0000012 0.0000017 0.0000010 0.0000000
Loading image data/three_28x28.pgm
Performing forward propagation ...
Resulting weights from Softmax:
0.0000000 0.0000000 0.0000000 0.9999288 0.0000000 0.0000711 0.0000000 0.0000000 0.0000000 0.0000000
Loading image data/five_28x28.pgm
Performing forward propagation ...
Resulting weights from Softmax:
0.0000000 0.0000008 0.0000000 0.0000002 0.0000000 0.9999820 0.0000154 0.0000000 0.0000012 0.0000006
Result of classification: 1 3 5
Test passed!

Testing half precision (math in single precision)
Loading image data/one_28x28.pgm
Performing forward propagation ...
Testing cudnnGetConvolutionForwardAlgorithm ...
Fastest algorithm is Algo 0
Testing cudnnFindConvolutionForwardAlgorithm ...
**** CUDNN_STATUS_SUCCESS for Algo 0: 0.012512 time requiring 0 memory
**** CUDNN_STATUS_SUCCESS for Algo 2: 0.027040 time requiring 28800 memory
**** CUDNN_STATUS_SUCCESS for Algo 1: 0.037120 time requiring 3464 memory
**** CUDNN_STATUS_SUCCESS for Algo 4: 0.060576 time requiring 207360 memory
**** CUDNN_STATUS_SUCCESS for Algo 7: 0.066880 time requiring 2057744 memory
Resulting weights from Softmax:
0.0000001 1.0000000 0.0000000 0.0000000 0.0000563 0.0000001 0.0000012 0.0000017 0.0000010 0.0000001
Loading image data/three_28x28.pgm
Performing forward propagation ...
Resulting weights from Softmax:
0.0000000 0.0000000 0.0000000 1.0000000 0.0000000 0.0000714 0.0000000 0.0000000 0.0000000 0.0000000
Loading image data/five_28x28.pgm
Performing forward propagation ...
Resulting weights from Softmax:
0.0000000 0.0000008 0.0000000 0.0000002 0.0000000 1.0000000 0.0000154 0.0000000 0.0000012 0.0000006
Result of classification: 1 3 5
Test passed!
```

Exhibit 7: Verification of cuDNN installation with MNIST example

Step 4: Installing TensorFlow 1.13

TensorFlow is “an open source machine learning framework for everyone”
(Source: [Google 2018](#))

Create a Virtual Environment (optional but recommended): Virtual environments help you to isolate installed Python packages from the

system. The following steps are adapted from TensorFlow installation guidelines which you can find [here](#).

To create a new virtual environment for Python3 and store it in `./venv`, execute the following:

```
$ virtualenv --system-site-packages -p python3 ./venv
```

Then, to activate the newly created environment, and also to activate it in the future, please execute:

```
$ source ./venv/bin/activate
```

Within the virtual environment, you can install packages without messing around with the host system itself. For example, you can upgrade pip and show the packages installed within the virtual environment by executing:

```
$ pip install --upgrade pip  
$ pip list
```

Later, you can exit the virtual environment by entering `$ deactivate` (DO NOT EXIT THE VIRTUAL ENVIRONMENT NOW!)

Install TensorFlow: Finally, you can install the GPU-enabled version of TensorFlow with the command:

```
$ pip install tf-nightly-gpu
```

IMPORTANT: Note that we are installing the latest tf-nightly-gpu build, since the non-nightly GPU-version of TensorFlow can't import "libcublas.so.9.0". I will update this once this issue is fixed.

Verify TensorFlow installation: Please execute the following Python snippet to verify your TensorFlow installation:

```
$ python -c "from tensorflow.python.client import device_lib; print(device_lib.list_local_devices())"
```

Note: If you get the error “ModuleNotFoundError: No module named ‘pandas’”, please install it via: `$ pip install pandas`

In case of a successfully installed GPU-enabled TensorFlow version, the output of the above will entail your GPU (in my case the GeForce RTX 2070) as shown below:

```
physical_device_desc: "device: 0, name: GeForce RTX 2070, pci bus id: 0000:07:00.0, compute capability: 7.5"
```

Troubleshooting

- **Issues with NVIDIA Display Driver after updating Linux kernel (December 27, 2018).** If you run into problems after upgrading your Linux kernel, please re-run step 1 of this tutorial to rebuild the kernel modules.
- **Errors when executing ./deviceQuery after successful building of CUDA samples (January 17th, 2019).** Please reboot your system and try again.
- **Random lags / lagging mouse after installing NVIDIA display driver.** Please sync your display device to one specific monitor by starting the nvidia-settings tool as root(`$ sudo nvidia-settings`). Here, navigate to “X Server XVideo Settings”. Now select one of your physical monitors to sync the display device to (instead of Auto).
- **Random system reboots while running complex TensorFlow models.** This could be due to spikes in energy consumption of your graphics card exceeding a maximum wattage threshold. Changing this threshold to a lower value can potentially mitigate this. See <https://github.com/tensorflow/tensorflow/issues/8858>.

Contributors

Danney Az, Z* (the superninja)

Epilog

I really hope this guide helps you to install TensorFlow with GPU-support on a freshly installed machine running Ubuntu 18.04 LTS with a CUDA 10 enabled NVIDIA GPU. Please let me know if you spot any bugs. I will do my best to incorporate them as fast as possible. Cheers!

For comparison purposes, here are my hardware specifications: NVIDIA GeForce RTX 2070 GPU, AMD Ryzen 5 2600X CPU, 32 GB DDR4 RAM, Samsung 512 GB NVMe m.2 SSD.