```
---
title: Installation: Ubuntu (step by step, including external libraries)
---
```

Ubuntu Installation

(step by step, including external libraries)

Guide for Ubuntu 14.04 with a 64 bit processor.

Tested on a Dell XPS laptop with a NVIDIA GeForce GT 525m and a Desktop PC with a NVIDIA GeForce GTX 580.

This guide is intended to be a (sort of) comprehensive step by step tutorial, including the installation of external dependencies.

This tutorial assumes some decisions about the versions and programs to install, to make it as straightforward to follow as possible, while still having as much advanced features as possible.

Notably:

- It uses **Anaconda Python** (which facilitates installation and has many great features).
- It uses **OpenBLAS** (which presumably increases performance over ATLAS).

The "code formatted" sections should be run in a terminal.

This guide was based on the guides in:

- Caffe: General, Caffe installation.
- Caffe: Ubuntu prerequisites.
- NVIDIA: CUDA Installation (more complete guide).
- Other resources as StackOverflow, Caffe-users group, forums, etc.

Install Git:

```
    sudo apt-get install git
```

NVIDIA drivers

- Find your graphics card model:

```
    lspci | grep -i nvidia
```

- Go to the NVIDIA website and find the latest driver version compatible with your graphics card, if you have a NVIDIA GeForce, go to: http://www.geforce.com/drivers
- Check the latest version compiled in "Proprietary GPU Drivers" PPA which also matches your graphics card in: https://launchpad.net/~graphics-drivers/+archive/ubuntu/ppa. For stability, you may want to use the recommended driver in that page.
- Add the "Proprietary GPU Drivers" PPA repository:

```
sudo add-apt-repository ppa:graphics-drivers/ppa
```

- Update the repositories database:

```
sudo apt-get update
```

- Install your selected version of the graphics card (this could break your graphic system, be sure to know how to use a shell to be able to repair it), for example:

```
sudo apt-get install nvidia-352
```

- Restart your system:

```
sudo shutdown -r now
```

- Check the NVIDIA drivers installed:

```
cat /proc/driver/nvidia/version
```

Fix flickering

If your screen is flickering, you can solve it with the following steps.

Note: the following may not apply to you.

- Install Compiz Config:

```
sudo apt-get install compizconfig-settings-manager
```

- From the launcher, execute CompizConfig Settings Manager
- Check the checkbox in *"Utility -> Workarounds -> Force full screen redraws (buffer swap) on repaint"*

CUDA

- Check that your computer is CUDA enabled (you should have a compatible NVIDIA graphics card):

```
lspci | grep -i nvidia
```

- You need to have an account or register in the NVIDIA developer program (it's free): https://developer.nvidia.com/user
- Download CUDA 7 from: https://developer.nvidia.com/cuda-downloads, you can download the "deb (network)" package (cuda repo).
- Go to the downloads directory:

```
cd ~/Downloads
```

- Install the package (the repository):

```
sudo dpkg -i cuda-repo-ubuntu1404*amd64.deb
```

- Update the repositories DB:

```
sudo apt-get update
```

- Install CUDA (it should list cuda packages):

```
sudo apt-get install cuda
```

- Add CUDA to the PATH variable (and make it permanent for the current user):

```
echo 'export PATH=/usr/local/cuda/bin:$PATH' >> ~/.bashrc
```

- Add CUDA to the LD_LIBRARY_PATH variable (and make it permanent for the current user):

```
echo 'export LD_LIBRARY_PATH=/usr/local/cuda/lib64:$LD_LIBRARY_PATH' >> ~/.bashrc
```

- Log out and log in again to activate the new variables (close and open the terminal).

Check CUDA installation

- To check the correct installation running the samples, install the samples in a directory under your home:

```
cuda-install-samples-7.0.sh ~/cuda-samples
```

- You can check the version of the NVIDIA drivers installed with:

```
cat /proc/driver/nvidia/version
```

- You can check the version of CUDA installed with:

```
nvcc -V
```

- Go to the samples directory:

```
cd ~/cuda-samples/NVIDIA*Samples
```

- Build the samples:

```
make -j $(($(nproc) + 1))
```

Note: you could just run `make`, but that last part (`-j $(($(nproc) + 1))`) executes the `make` command in parallel, using the number of cores in your machine (plus one), so the compilation would finish faster.

- Restart your computer:

```
sudo shutdown -r now
```

- Check the CUDA installation, run:

```
bin/x86_64/linux/release/deviceQuery
```

- Check that it detects a CUDA device.
- Check that it detects your graphics card.
- Check at the end that it says that the test passed.
- Run the bandwidth test to check that the system and the CUDA device are capable to communicate:

```
bin/x86_64/linux/release/bandwidthTest
```

Install OpenBLAS

Note: you could just use "ATLAS" with: `sudo apt-get install libatlas-base-dev`. But you would probably get a better performance from OpenBLAS following the next instructions, and it's quite straightforward.

- Install Fortran compiler:

```
sudo apt-get install gfortran
```

- Create a "code" directory to store your code:

Note: you can put your code wherever you want, so, modify this part as you wish, but I will assume that your are doing it.

```
mkdir ~/code
```

- Enter that directory:

```
cd ~/code
```

- Clone OpenBLAS:

```
 git clone https://github.com/xianyi/OpenBLAS.git
```

- Go to the OpenBLAS directory:

```
cd OpenBLAS
```

- Compile OpenBLAS:

```
make -j $(($(nproc) + 1))
```

Note: you could just run `make`, but that last part (`-j $(($(nproc) + 1))`) executes the `make` command in parallel, using the number of cores in your machine (plus one), so the compilation would finish faster.

- Install OpenBLAS:

```
sudo make PREFIX=/usr/local install
```

Install Boost

```
sudo apt-get install libboost-all-dev
```

Install OpenCV

```
sudo apt-get install libopencv-dev
```

Install: protobuf, glog, gflags

```
sudo apt-get install libprotobuf-dev libgoogle-glog-dev libgflags-dev protobuf-compiler
```

Install IO libraries: hdf5, leveldb, snappy, lmdb

```
sudo apt-get install libhdf5-serial-dev libleveldb-dev libsnappy-dev liblmdb-dev
```

Install Anaconda Python

- Go to the Anaconda website and download Anaconda for Linux (you would probably want the 64x version): http://continuum.io/downloads
- Go to the downloads directory:

  ```
  cd ~/Downloads
  ```

- Execute the installer:

  ```
  bash Anaconda*.sh
  ```

- When asked if add Anaconda to the PATH for the current user, type "yes".
- Log out and log in again to activate the new variables (close and open the terminal).

Install HDF5

- Install HDF5 with conda, although it will only be used by Anaconda:

  ```
  conda install hdf5
  ```

Configure the HDF5 version

Note: this is a quick fix for a minor bug / issue with the version of libhdf5. If you know a better / proper way to solve it, let me know. If this section doesn't apply to you, omit it.

- Go to the libraries directory:

  ```
  cd /usr/lib/x86_64-linux-gnu
  ```

- Link the system version of HDF5 from 7 to 9:

  ```
  sudo ln -s libhdf5.so.7 libhdf5.so.9
  sudo ln -s libhdf5_hl.so.7 libhdf5_hl.so.9
  ```

- Update the "Dynamic Linker":

  ```
  sudo ldconfig
  ```

Install CuDNN

Note: for this to work, you need a NVIDIA GPU based on the *"Kepler"* architeture (or the newer, *"Maxwell"* architecture).

Note: although this guide includes all the build, installation and configuration of CuDNN, only the build and installation was tested, the usage hasn't been tested. (It was only tested on a NVIDIA GT 525m and a NVIDIA GTX 580 which have a *"Fermi"* (older) architecture, and is not supported by CuDNN.

- You need to have an account or register in the NVIDIA developer program (it's free): https://developer.nvidia.com/user
- Using your account, download CuDNN: https://developer.nvidia.com/cuDNN

- Go to the downloads directory:

```
cd ~/Downloads/
```

- Uncompress the CuDNN download:

```
tar xvf cudnn*.tgz
```

- Enter the uncompressed directory:

```
cd cuda
```

- Copy the *.h files to your CUDA installation:

```
sudo cp */*.h /usr/local/cuda/include/
```

- Copy the *.so* files to your CUDA installation:

```
sudo cp */*.so* /usr/local/cuda/lib64/
```

Configure and compile Caffe

- Create a "code" directory to store your code:

Note: you can put your code wherever you want, so, modify this part as you wish, but I will assume that your are following the tutorial.

```
mkdir ~/code
```

- Enter that directory:

  cd ~/code

- Clone Caffe

```
git clone https://github.com/BVLC/caffe.git
```

- Enter the new Caffe directory

```
cd caffe
```

- Copy the Makefile.config

```
cp Makefile.config.example Makefile.config
```

- Enable CuDNN, set the USE_CUDNN := 1 flag in Makefile.config:

Note: this will only work with a GPU with Kepler or newer architecture (Maxwell). This tutorial was tested on an older GPU, so this part was omitted.

```
sed -i 's/# USE_CUDNN := 1/USE_CUDNN := 1/' Makefile.config
```

- Enable OpenBLAS:

Note: if you didn't install OpenBLAS and installed ATLAS, omit this part.

```
sed -i 's/BLAS := atlas/BLAS := open/' Makefile.config
```

- Enable Anaconda:

```
sed -i 's|# ANACONDA_HOME := $(HOME)/anaconda|ANACONDA_HOME := $(HOME)/anaconda|' Makefile.config
sed -i 's|# PYTHON_INCLUDE := $(ANACONDA_HOME)|PYTHON_INCLUDE := $(ANACONDA_HOME)|' Makefile.config
sed -i 's|# $(ANACONDA_HOME)|$(ANACONDA_HOME)|' Makefile.config
sed -i 's|# PYTHON_LIB := $(ANACONDA_HOME)|PYTHON_LIB := $(ANACONDA_HOME)|' Makefile.config
sed -i 's|# WITH_PYTHON_LAYER := 1|WITH_PYTHON_LAYER := 1|' Makefile.config
```

- Install Python dependencies using conda:

```
for req in $(cat python/requirements.txt); do conda install $req; done
```

- Install Python dependencies using pip (to install what couldn't be installed by conda):

```
pip install -r python/requirements.txt
```

- Remove bad Anaconda libm:

Note: this is a fix for a common bug with Anaconda. It may not apply to you.

```
mkdir ~/anaconda/lib/libm.orig.d
mv ~/anaconda/lib/libm.* ~/anaconda/lib/libm.orig.d
```

- Build Caffe:

```
make all -j $(($(nproc) + 1))
```

- Build the tests:

```
make test -j $(($(nproc) + 1))
```

- Run the tests:

```
make runtest -j $(($(nproc) + 1))
```

- After running `make runtest`, check that all the tests passed.
- Build PyCaffe (Python interface):

```
make pycaffe -j $(($(nproc) + 1))
```

Note: you could just run `make all` or `make test` or `make runtest` or `make pycaffe` respectively, but that last part (`-j $(($(nproc) + 1))`) executes the `make` command in parallel, using the number of cores in your machine (plus one), so the compilation would finish faster.

- Export a variable CAFFE_ROOT with your Caffe root directory and make it permanent:

```
echo "export CAFFE_ROOT=$(pwd)" >> ~/.bashrc
```

- Add Caffe to your PYTHONPATH variable using your CAFFE_ROOT:

Technical Note: you could add Caffe to your PYTHONPATH directly, but by using a CAFFE_ROOT variable, you could move your Caffe directory and would only have to modify your CAFFE_ROOT variable. And you could use the same to export Caffe binaries to your PATH variable.

```
echo 'export PYTHONPATH=$CAFFE_ROOT/python:$PYTHONPATH' >> ~/.bashrc
```

- Log out and log in again to activate the new variables (close and open the terminal).
- To test that PyCaffe is working open IPython:

  ```
  ipython
  ```

- Import "caffe":

  ```
  import caffe
  ```

- Check that there are no errors.

Congratulations, you have an up and running Caffe installation.

Now you may want to follow one of the tutorials:

- LeNet MNIST Tutorial
- ImageNet tutorial