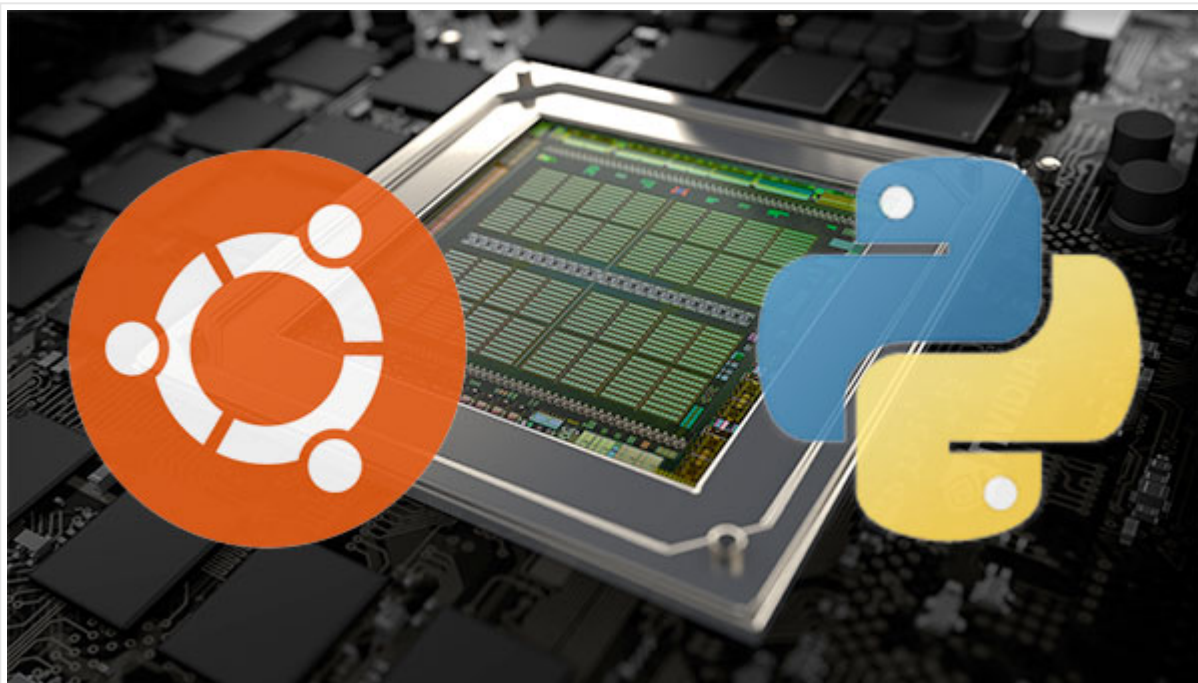


# Setting up Ubuntu 16.04 + CUDA + GPU for deep learning with Python

by Adrian Rosebrock on September 27, 2017 in [Deep Learning](#), [DL4CV](#)



Welcome back! This is the fourth post in the deep learning development environment configuration series which accompany my new book, [Deep Learning for Computer Vision with Python](#).

Today, we will configure Ubuntu + NVIDIA GPU + CUDA with everything you need to be successful when training your own deep learning networks on your GPU.

Links to related tutorials can be found here:

- [Your deep learning + Python Ubuntu virtual machine](#)
- [Pre-configured Amazon AWS deep learning AMI with Python](#)
- [Configuring Ubuntu for deep learning with Python](#) (for a CPU *only* environment)
- Setting up Ubuntu 16.04 + CUDA + GPU for deep learning with Python (this post)
- Configuring macOS for deep learning with Python (releasing on Friday)

If you have an NVIDIA CUDA compatible GPU, you can use this tutorial to configure your deep learning development to train and execute neural networks on your optimized GPU hardware.

**Let's go ahead and get started!**

## Setting up Ubuntu 16.04 + CUDA + GPU for deep learning with Python

If you've reached this point, you are likely serious about deep learning and want to train your neural networks with a GPU.

Graphics Processing Units are great at deep learning for their parallel processing architecture — in fact, these days there are many GPUs built *specifically* for deep learning — they are put to use outside the domain of computer gaming.

NVIDIA is the market leader in deep learning hardware, and quite frankly the primary option I recommend if you are getting in this space. It is worth getting familiar with their lineup of products (hardware and software) so you know what you're paying for if you're using an instance in the cloud or building a machine yourself. Be sure to check out [this developer page](#).

It is common to share high end GPU machines at universities and companies. Alternatively, you may build one, buy one ([as I did](#)), or rent one in the cloud (as I still do today).

If you are just doing a couple experiments then using a cloud service provider such as Amazon, Google, or FloydHub for a time-based usage charge is the way to go.

Longer term if you are working on deep learning experiments daily, then it would be wise to have one on hand for cost savings purposes (assuming you're willing to keep the hardware and software updated regularly).

**Note:** For those utilizing **AWS's EC2**, I recommend you select the **p2.xlarge**, **p2.8xlarge**, or **p2.16xlarge** machines for compatibility with these instructions (depending on your use case scenario and budget). The older instances, **g2.2xlarge** and **g2.8xlarge** are not compatible with the version of CUDA and cuDNN in this tutorial. I also recommend that you have **about 32GB of space on your OS drive/partition**. 16GB didn't cut it for me on my EC2 instance.

It is important to point out that **you don't need access to an expensive GPU machine to get started with Deep Learning**. Most modern laptop CPUs will do just fine with the small experiments presented in the early chapters in my book. As I say, "fundamentals before funds" — meaning, get acclimated with modern deep learning fundamentals and concepts before you bite off more than you can chew with expensive hardware and cloud bills. My book will allow you to do just that.

## How hard is it to configure Ubuntu with GPU support for deep learning?

You'll soon find out below that configuring a GPU machine isn't a cakewalk. In fact there are quite a few steps and potential for things to go sour. That's why I have built a custom Amazon Machine Instance (AMI) pre-configured and pre-installed for the community to accompany my book.

I detailed how to get it loaded into your AWS account and how to boot it up in [this previous post](#).

Using the AMI is *by far* the fastest way to get started with deep learning on a GPU. Even if you *do* have a GPU, it's worth experimenting in the Amazon EC2 cloud so you can tear down an instance (if you make a mistake) and then immediately boot up a new, fresh one.

Configuring an environment on your own is directly related to your:

1. Experience with Linux
2. Attention to detail
3. Patience.

First, you must be very comfortable with the command line.

Many of the steps below have commands that you can simply copy and paste into your terminal; however it is important that you read the output, note any errors, try to resolve them prior to moving on to the next step.

You *must* pay particular attention to the order of the instructions in this tutorial, and furthermore pay attention to the commands themselves.

I actually do recommend copying and pasting to make sure you don't mess up a command (in one case below backticks versus quotes could get you stuck).

If you're up for the challenge, then I'll be right there with you getting your environment ready. In fact I encourage you to leave comments so that the PyImageSearch community can offer you assistance. Before you leave a comment be sure to review the **post** and **comments** to make sure you didn't leave a step out.

Without further ado, let's get our hands dirty and walk through the configuration steps.

## Step #0: Turn off X server/X window system

Before we get started I need to point out an ***important prerequisite***. You need to perform one of the following prior to following the instructions below:

1. SSH into your GPU instance (with X server *off/disabled*).
2. Work directly on your GPU machine without your X server running (the X server, also known as X11, is your graphical user interface on the desktop). I suggest you try one of the methods outlined on [this thread](#).

There are a few methods to accomplish this, some easy and others a bit more involved.

The **first method** is a bit of a hack, but it works:

1. Turn off your machine.
2. Unplug your monitor.
3. Reboot.
4. SSH into your machine from a separate system.
5. Perform the install instructions.

This approach works great and is by far the easiest method. By unplugging your monitor X server will not automatically start. From there you can SSH into your machine from a separate computer and follow the instructions outline in this post.

The **second method** assumes you have already booted the machine you want to configure for deep learning:

1. Close all running applications.
2. Press `ctrl + alt + F2`.
3. Login with your username and password.
4. Stop X server by executing `sudo service lightdm stop`.
5. Perform the install instructions.

Please note that you'll need a separate computer next to you to read the instructions or execute the commands. Alternatively you could use a text-based web browser.

## Step #1: Install Ubuntu system dependencies

Now that we're ready, let's get our Ubuntu OS up to date:

```
Setting up Ubuntu 16.04 + CUDA + GPU for deep learning with Python 2.7 and 3.5  
1 $ sudo apt-get update  
2 $ sudo apt-get upgrade
```

Then, let's install some necessary development tools, image/video I/O, GUI operations and various other packages:

```
Setting up Ubuntu 16.04 + CUDA + GPU for deep learning with Python 2.7 and 3.5  
1 $ sudo apt-get install build-essential cmake git unzip pkg-config  
2 $ sudo apt-get install libjpeg-dev libtiff5-dev libjasper-dev lib  
3 $ sudo apt-get install libavcodec-dev libavformat-dev libswscale-  
4 $ sudo apt-get install libxvidcore-dev libx264-dev  
5 $ sudo apt-get install libgtk-3-dev  
6 $ sudo apt-get install libhdf5-serial-dev graphviz  
7 $ sudo apt-get install libopenblas-dev libatlas-base-dev gfortran  
8 $ sudo apt-get install python-tk python3-tk python-imaging-tk
```

Next, let's install both Python 2.7 and Python 3 header files so that we can compile OpenCV with Python bindings:

```
Setting up Ubuntu 16.04 + CUDA + GPU for deep learning with Python 2.7 and 3.5  
1 $ sudo apt-get install python2.7-dev python3-dev
```

We also need to prepare our system to swap out the default drivers with NVIDIA CUDA drivers:

```
Setting up Ubuntu 16.04 + CUDA + GPU for deep learning with Python 2.7 and 3.5  
1 $ sudo apt-get install linux-image-generic linux-image-extra-virt  
2 $ sudo apt-get install linux-source linux-headers-generic
```

That's it for Step #1, so let's continue on.

## Step #2: Install CUDA Toolkit

The CUDA Toolkit installation step requires attention to detail for it to go smoothly.

First disable the Nouveau kernel driver by creating a new file:

```
Setting up Ubuntu 16.04 + CUDA + GPU for deep learning with Python - PyImageSearch  
1 $ sudo nano /etc/modprobe.d/blacklist-nouveau.conf
```

Feel free to use your favorite terminal text editor such as `vim` or `emacs` instead of `nano`.

Add the following lines and then save and exit:

```
Setting up Ubuntu 16.04 + CUDA + GPU for deep learning with Python - PyImageSearch  
1 blacklist nouveau  
2 blacklist lbm-nouveau  
3 options nouveau modeset=0  
4 alias nouveau off  
5 alias lbm-nouveau off
```

Your session should look like the following (if you are using nano):

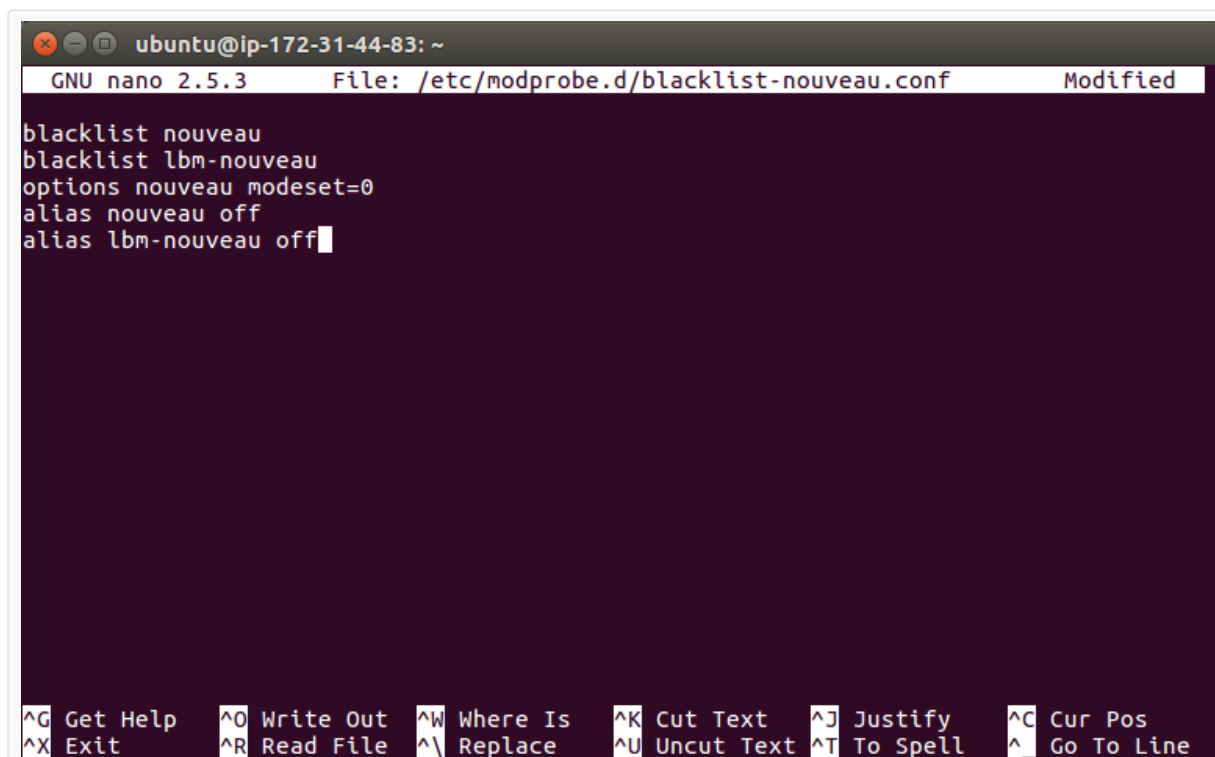


Figure 1: Editing the `blacklist-nouveau.conf` file with the `nano` text editor.

Next let's update the initial RAM filesystem and reboot the machine:

```
Setting up Ubuntu 16.04 + CUDA + GPU for deep learning with Python - PyImageSearch  
1 $ echo options nouveau modeset=0 | sudo tee -a /etc/modprobe.d/nouveau.conf  
2 $ sudo update-initramfs -u  
3 $ sudo reboot
```

You will lose your SSH connection at the reboot step, so wait patiently and then reconnect before moving on.

You will want to download the CUDA Toolkit v8.0 via the NVIDIA CUDA Toolkit website:

<https://developer.nvidia.com/cuda-80-ga2-download-archive>

Once you're on the download page, select `Linux => x86_64 => Ubuntu => 16.04 => runfile (local)`.

Here is a screenshot of the download page:

**Select Target Platform** ⓘ

Click on the green buttons that describe your target platform. Only supported platforms will be shown.

**Operating System** Windows Linux Mac OSX

**Architecture** ⓘ x86\_64 ppc64le

**Distribution** Fedora OpenSUSE RHEL CentOS SLES Ubuntu

**Version** 16.04 14.04

**Installer Type** ⓘ runfile [local] deb [local] deb [network] cluster [local]

**Download Installers for Linux Ubuntu 16.04 x86\_64**

The base installer is available for download below.  
There is 1 patch available. This patch requires the base installer to be installed first.

**> Base Installer** Download [1.4 GB] ⬇

Installation Instructions:

1. Run ``sudo sh cuda_8.0.61_375.26_linux.run``
2. Follow the command-line prompts

**> Patch 2 [Released Jun 26, 2017]** Download [95.3 MB] ⬇

cuBLAS Patch Update to CUDA 8: Includes performance enhancements and bug-fixes

The CUDA Toolkit contains Open-Source Software. The source code can be found [here](#).  
The checksums for the installer and patches can be found in [Installer Checksums](#).  
For further information, see the [Installation Guide for Linux](#) and the [CUDA Quick Start Guide](#).

**Figure 2:** The CUDA Toolkit download page.

From there, download the `-run` file which should have the filename `cuda_8.0.61_375.26_linux-run` or similar. To do this, simply right-click to copy the download link and use `wget` on your remote GPU box:

```
Setting up Ubuntu 16.04 + CUDA + GPU for deep learning with Python
1 $ wget https://developer.nvidia.com/compute/cuda/8.0/Prod2/local_
```

**Important:** At the time of this writing there is a minor discrepancy on the NVIDIA website. As shown in **Figure 2** under the “Base Installer” download, the filename (as is written) ends with `.run`. The actual downloadable file ends with `-run`. You should be good to go in copying my `wget` + URL command for now unless NVIDIA changes the filename again.

**Note:** You will need to click the “<=>” button in the code block toolbar above to expand the code block. This will enable you to copy the **full URL** to the `-run` file.

From there, unpack the `-run` file:

```
Setting up Ubuntu 16.04 + CUDA + GPU for deep learning with Python
1 $ chmod +x cuda_8.0.61_375.26_linux-run
2 $ mkdir installers
3 $ sudo ./cuda_8.0.61_375.26_linux-run -extract=`pwd`/installers
```



The last step in the block above can take 30-60 seconds depending on the speed of your machine.

Now it is time to install the NVIDIA kernel driver:

```
Setting up Ubuntu 16.04 + CUDA + GPU for deep learning with Python - PyImageSearch
1 $ cd installers
2 $ sudo ./NVIDIA-Linux-x86_64-375.26.run
```

During this process, accept the license and follow prompts on the screen.

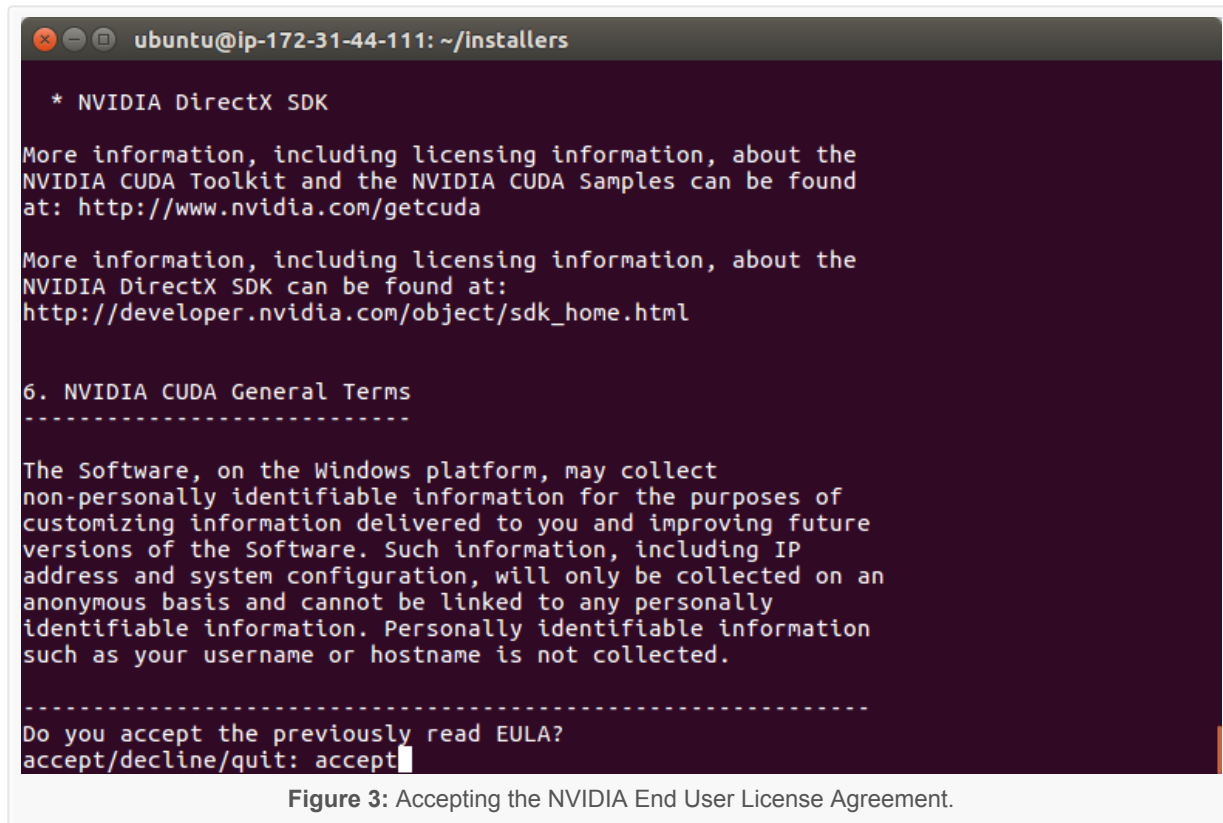


Figure 3: Accepting the NVIDIA End User License Agreement.

From there, add the NVIDIA loadable kernel module (LKM) to the Linux kernel:

```
Setting up Ubuntu 16.04 + CUDA + GPU for deep learning with Python - PyImageSearch
1 $ modprobe nvidia
```

Install the CUDA Toolkit and examples:

```
Setting up Ubuntu 16.04 + CUDA + GPU for deep learning with Python - PyImageSearch
1 $ sudo ./cuda-linux64-rel-8.0.61-21551265.run
2 $ sudo ./cuda-samples-linux-8.0.61-21551265.run
```

Again, accepting the licenses and following the default prompts. You may have to press 'space' to scroll through the license agreement and then enter "accept" as I've done in the image above. When it asks you for installation paths, just press `<enter>` to accept the defaults.

Now that the NVIDIA CUDA driver and tools are installed, you need to update your `~/.bashrc` file to include CUDA Toolkit (I suggest using terminal text editors such as `vim`, `emacs`, or `nano`):

```
Setting up Ubuntu 16.04 + CUDA + GPU for deep learning with Python - PyImageSearch
```

```

1 # NVIDIA CUDA Toolkit
2 export PATH=/usr/local/cuda-8.0/bin:$PATH
3 export LD_LIBRARY_PATH=/usr/local/cuda-8.0/lib64/

```

Now, reload your `~/.bashrc` ( `source ~/.bashrc` ) and then test the CUDA Toolkit installation by compiling the `deviceQuery` example program and running it:

```

Setting up Ubuntu 16.04 + CUDA + GPU for deep learning with Python
1 $ source ~/.bashrc
2 $ cd /usr/local/cuda-8.0/samples/1_Utilities/deviceQuery
3 $ sudo make
4 $ ./deviceQuery
5 deviceQuery, CUDA Driver = CUDART, CUDA Driver Version = 8.0, CUDA
6 Result = PASS

```

**Note:** Calling `source` on `~/.bashrc` only has to be done **once** for our current shell session. Anytime we open up a new terminal, the contents of `~/.bashrc` will be **automatically** executed (including our updates).

At this point if you have a `Result = PASS`, then *congratulations* because you are ready to move on to the next step.

If you do not see this result, I suggest you repeat Step #2 and examine the output of each and every command *carefully* to ensure there wasn't an error during the install.

## Step #3: Install cuDNN (CUDA Deep Learning Neural Network library)

For this step, you will need to [Create a free account with NVIDIA](#) and [download cuDNN](#).

For this tutorial I used **cuDNN v6.0** for Linux *which is what TensorFlow requires*.

Due to NVIDIA's *required authentication to access the download*, you may not be able to use `wget` on your remote machine for the download.

Instead, download the file to your *local machine* and then (on your local machine) use `scp` (Secure Copy) while replacing `<username>` and `<password>` with appropriate values to update the file to your remote instance (again, assuming you're accessing your machine via SSH):

```

Setting up Ubuntu 16.04 + CUDA + GPU for deep learning with Python
1 scp -i EC2KeyPair.pem ~/Downloads/cudnn-8.0-linux-x64-v6.0.tgz \
2 username@your_ip_address:~

```

Next, untar the file and then copy the resulting files into `lib64` and `include` respectively, using the `-P` switch to preserve sym-links:

```

Setting up Ubuntu 16.04 + CUDA + GPU for deep learning with Python
1 $ cd ~
2 $ tar -zxf cudnn-8.0-linux-x64-v6.0.tgz
3 $ cd cuda
4 $ sudo cp -P lib64/* /usr/local/cuda/lib64/
5 $ sudo cp -P include/* /usr/local/cuda/include/
6 $ cd ~

```

That's it for **Step #3** — there isn't much that can go wrong here, so you should be ready to proceed.



## Step #4: Create your Python virtual environment

In this section we will get a Python virtual environment configured on your system.

### Installing pip

The first step is to install `pip`, a Python package manager:

```
Setting up Ubuntu 16.04 + CUDA + GPU for deep learning with Python - PyImageSearch ythohl
1 $ wget https://bootstrap.pypa.io/get-pip.py
2 $ sudo python get-pip.py
3 $ sudo python3 get-pip.py
```

### Installing virtualenv and virtualenvwrapper

Using `pip`, we can install any package in the Python Package Index quite easily including `virtualenv` and `virtualenvwrapper`. As you know, I'm a fan of Python virtual environments and I encourage you to use them for deep learning as well.

In case you have multiple projects on your machine, using virtual environments will allow you to isolate them and install different versions of packages. In short, using both `virtualenv` and `virtualenvwrapper` allow you to solve the “*Project X depends on version 1.x, but Project Y needs 4.x*” dilemma.

The folks over at RealPython may be able to convince you if I haven't, so [give this excellent blog post on RealPython a read](#).

Again, let me reiterate that it's **standard practice** in the Python community to be leveraging virtual environments of some sort, so I suggest you do the same:

```
Setting up Ubuntu 16.04 + CUDA + GPU for deep learning with Python - PyImageSearch ythohl
1 $ sudo pip install virtualenv virtualenvwrapper
2 $ sudo rm -rf ~/.cache/pip get-pip.py
```

Once we have `virtualenv` and `virtualenvwrapper` installed, we need to update our `~/.bashrc` file to include the following lines at the *bottom* of the file:

```
Setting up Ubuntu 16.04 + CUDA + GPU for deep learning with Python - PyImageSearch ythohl
1 # virtualenv and virtualenvwrapper
2 export WORKON_HOME=$HOME/.virtualenvs
3 export VIRTUALENVWRAPPER_PYTHON=/usr/bin/python3
4 source /usr/local/bin/virtualenvwrapper.sh
```

After editing our `~/.bashrc` file, we need to reload the changes:

```
Setting up Ubuntu 16.04 + CUDA + GPU for deep learning with Python - PyImageSearch ythohl
1 $ source ~/.bashrc
```

Now that we have installed `virtualenv` and `virtualenvwrapper`, the next step is to actually *create* the Python virtual environment — we do this using the `mkvirtualenv` command.

### Creating the dl4cv virtual environment

In past install tutorials, I've presented the choice of Python 2.7 or Python 3. At this point in the Python 3 development cycle, I consider it stable and the right choice. You may elect to use Python 2.7 if you have specific compatibility requirements, but **for the purposes of my book we will use Python 3**.

With that said, for the following command, ensure you set the `-p` flag to `python3`.

```
Setting up Ubuntu 16.04 + CUDA + GPU for dee... ythonl.  
1 $ mkvirtualenv dl4cv -p python3
```

You can name this virtual environment whatever you like (and create as many Python virtual environments as you want), but for the time being, I would suggest sticking with the `dl4cv` name as that is what I'll be using throughout the rest of this tutorial.

## Verifying that you are in the “dl4cv” virtual environment

If you ever reboot your Ubuntu system; log out and log back in; or open up a new terminal, you'll need to use the `workon` command to re-access your `dl4cv` virtual environment. An example of the `workon` command follows:

```
Setting up Ubuntu 16.04 + CUDA + GPU for dee... ythonl.  
1 $ workon dl4cv
```

To validate that you are in the `dl4cv` virtual environment, simply examine your command line — *if you see the text `(dl4cv)` preceding your prompt, then you **are** in the `dl4cv` virtual environment:*



Figure 4: Inside the `dl4cv` virtual environment.

Otherwise if you **do not** see the `dl4cv` text, then you **are not** in the `dl4cv` virtual environment:

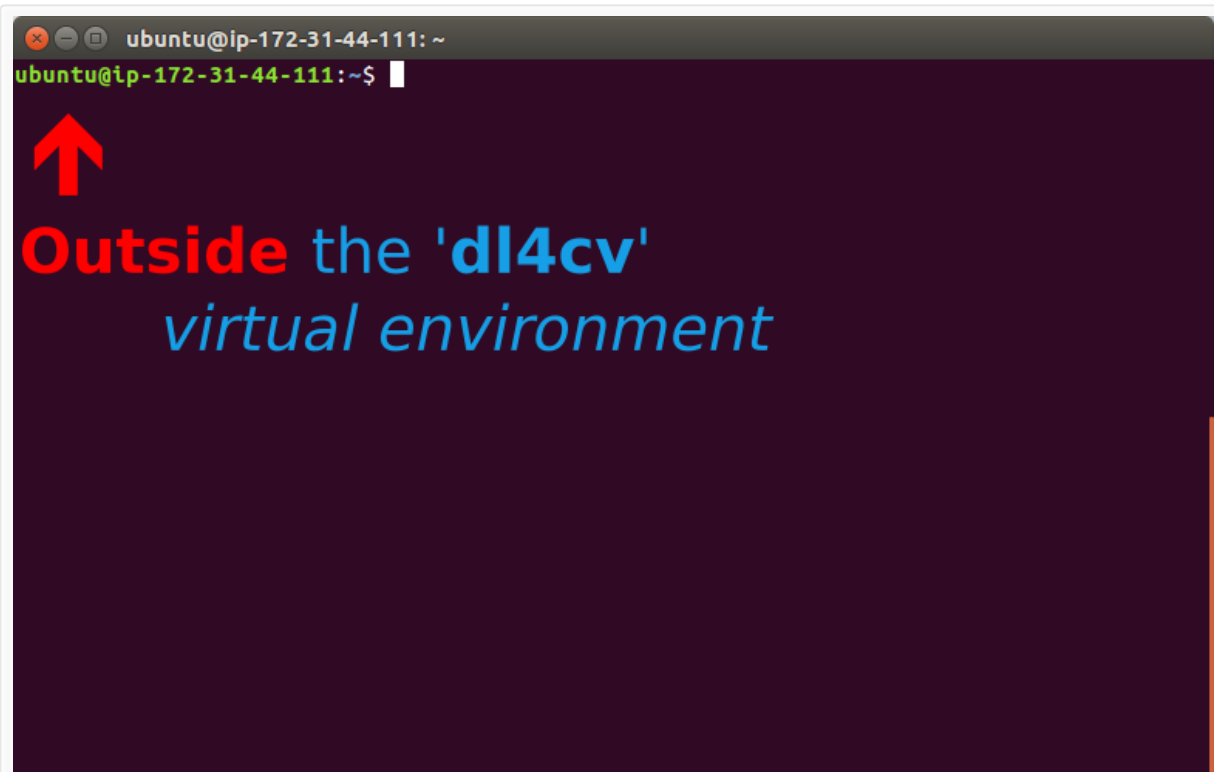


Figure 5: Outside the *dl4cv* virtual environment. Execute *workon dl4cv* to activate the environment.

## Installing NumPy

The final step before we compile OpenCV is to install [NumPy](#), a Python package used for numerical processing. To install NumPy, ensure you are in the `dl4cv` virtual environment (otherwise NumPy will be installed into the *system* version of Python rather than the `dl4cv` environment).

From there execute the following command:

```
Setting up Ubuntu 16.04 + CUDA + GPU for deep learning with Python 3.5.2 | ython |
1 $ pip install numpy
```

Once NumPy is installed in your virtual environment, we can move on to compile and install OpenCV.

## Step #5: Compile and Install OpenCV

First you'll need to download [opencv](#) and [opencv\\_contrib](#) into your home directory. For this install guide, we'll be using OpenCV 3.3:

```
Setting up Ubuntu 16.04 + CUDA + GPU for deep learning with Python 3.5.2 | ython |
1 $ cd ~
2 $ wget -O opencv.zip https://github.com/Itseez/opencv/archive/3.3
3 $ wget -O opencv_contrib.zip https://github.com/Itseez/opencv_con
```

Then, unzip both files:

```
Setting up Ubuntu 16.04 + CUDA + GPU for deep learning with Python 3.5.2 | ython |
1 $ unzip opencv.zip
2 $ unzip opencv_contrib.zip
```

## Running CMake

In this step we create a build directory and then run CMake:

```
Setting up Ubuntu 16.04 + CUDA + GPU for deep learning with Python 3
1 $ cd ~/opencv-3.3.0/
2 $ mkdir build
3 $ cd build
4 $ cmake -D CMAKE_BUILD_TYPE=RELEASE \
5     -D CMAKE_INSTALL_PREFIX=/usr/local \
6     -D WITH_CUDA=OFF \
7     -D INSTALL_PYTHON_EXAMPLES=ON \
8     -D OPENCV_EXTRA_MODULES_PATH=~/opencv_contrib-3.3.0/modules \
9     -D BUILD_EXAMPLES=ON ..
```

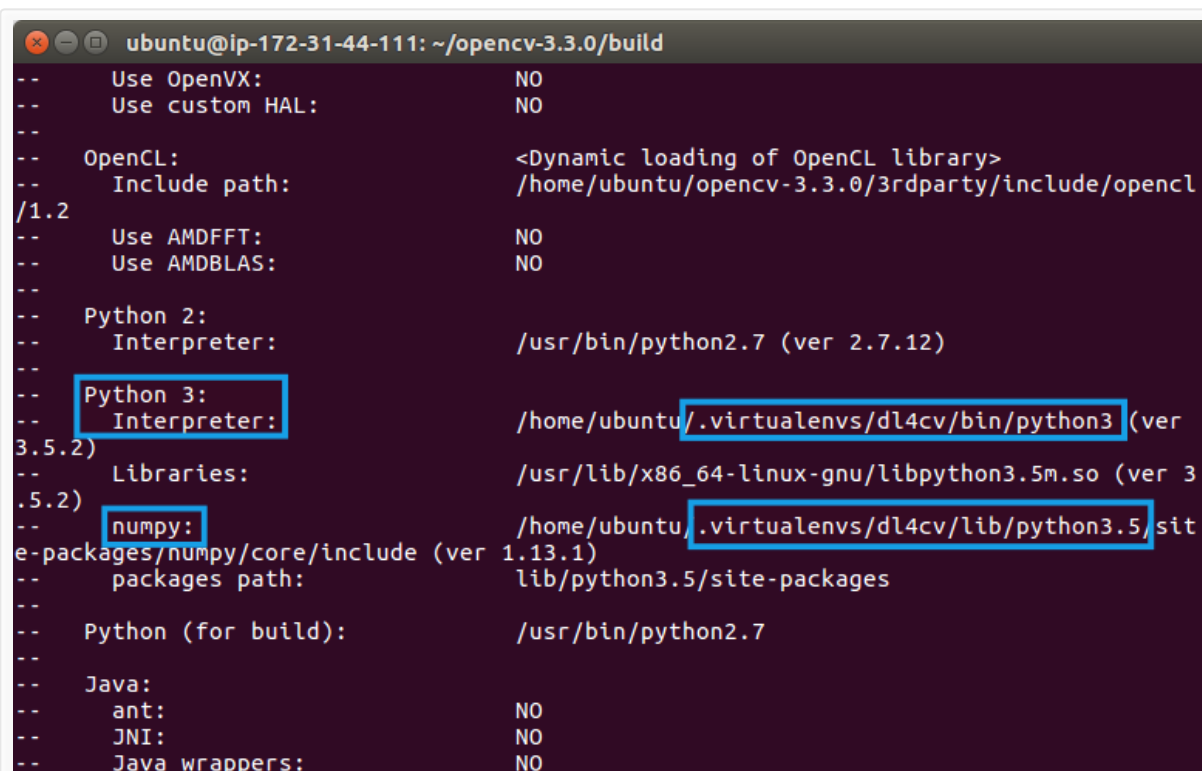
**Note:** I turned CUDA off as it can lead to compile errors on some machines. The CUDA optimizations would internally be used for C++ functions so it doesn't make much of a difference with Python + OpenCV. Again, the primary use of CUDA in this blog post is to optimize our deep learning libraries, not OpenCV itself.

For CMake, it is important that your flags match mine for compatibility. Also, make sure that your `opencv_contrib` version is the exact same as the `opencv` version you downloaded (in this case version `3.3.0`).

Before we move on to the actual compilation step, *make sure you examine the output of CMake*.

Start by scrolling to the section titled `Python 3`.

Make sure that your Python 3 section looks like the figure below:



```
ubuntu@ip-172-31-44-111: ~/opencv-3.3.0/build
-- Use OpenVX: NO
-- Use custom HAL: NO
--
-- OpenCL:
--   Include path: /home/ubuntu/opencv-3.3.0/3rdparty/include/opencvcl
/1.2
-- Use AMDFFT: NO
-- Use AMDBLAS: NO
--
-- Python 2:
--   Interpreter: /usr/bin/python2.7 (ver 2.7.12)
--
-- Python 3:
--   Interpreter: /home/ubuntu/.virtualenvs/dl4cv/bin/python3 (ver
3.5.2)
--   Libraries: /usr/lib/x86_64-linux-gnu/libpython3.5m.so (ver 3
.5.2)
--   numpy: /home/ubuntu/.virtualenvs/dl4cv/lib/python3.5/site
e-packages/numpy/core/include (ver 1.13.1)
--   packages path: lib/python3.5/site-packages
--
-- Python (for build): /usr/bin/python2.7
--
-- Java:
--   ant: NO
--   JNI: NO
--   Java wrappers: NO
```

**Figure 6:** Verifying that CMake has properly set up the compile to use the correct Python 3 Interpreter and version of NumPy. Both Python 3 and NumPy should be pulled from the `dl4cv` virtual environment.

Ensure that the Interpreter points to our `python3.5` binary located in the `dl4cv` virtual environment while `numpy` points to our NumPy install.

In either case if you **do not** see the `dl4cv` virtual environment in these variables' paths, then ***it's almost certainly because you are NOT in the `dl4cv` virtual environment prior to running CMake!***

If this is the case, access the `dl4cv` virtual environment using `workondl4cv` and re-run the command outlined above.

## Compiling OpenCV

Now we are now ready to compile OpenCV :

```
Setting up Ubuntu 16.04 + CUDA + GPU for deep learning with Python 3.5
1 $ make -j4
```

**Note:** If you run into compilation errors, you may run the command `makeclean` and then just compile without the flag: `make` . You can adjust the number of processor cores you use the compile OpenCV via the `-j` switch (in the example above, I'm compiling OpenCV with four cores).

From there, all you need to do is to install OpenCV 3.3:

```
Setting up Ubuntu 16.04 + CUDA + GPU for deep learning with Python 3.5
1 $ sudo make install
2 $ sudo ldconfig
3 $ cd ~
```

You can also delete your `opencv` and `opencv_contrib` directories to free up space on your system; however, I *highly recommend* that you wait until the end of this tutorial and ensured OpenCV has been correctly installed *before* you delete these files (otherwise you'll have to download them again).

## Symbolic linking OpenCV to your virtual environment

To sym-link our OpenCV bindings into the `dl4cv` virtual environment, issue the following commands

```
Setting up Ubuntu 16.04 + CUDA + GPU for deep learning with Python 3.5
1 $ cd ~/.virtualenvs/dl4cv/lib/python3.5/site-packages/
2 $ ln -s /usr/local/lib/python3.5/site-packages/cv2.cpython-35m-x86_64-linux-gnu.so .
3 $ cd ~
```

**Note:** Make sure you click “<=>” button in the toolbar above to expand the code block. From there, ensure you copy and paste the `ln` command correctly, otherwise you'll create an invalid sym-link and Python will not be able to find your OpenCV bindings.

Your `.so` file may be some variant of what is shown above, so be sure to use the appropriate file.

## Testing your OpenCV 3.3 install

Now that we've got OpenCV 3.3 installed and linked, let's do a quick sanity test to see if things work:

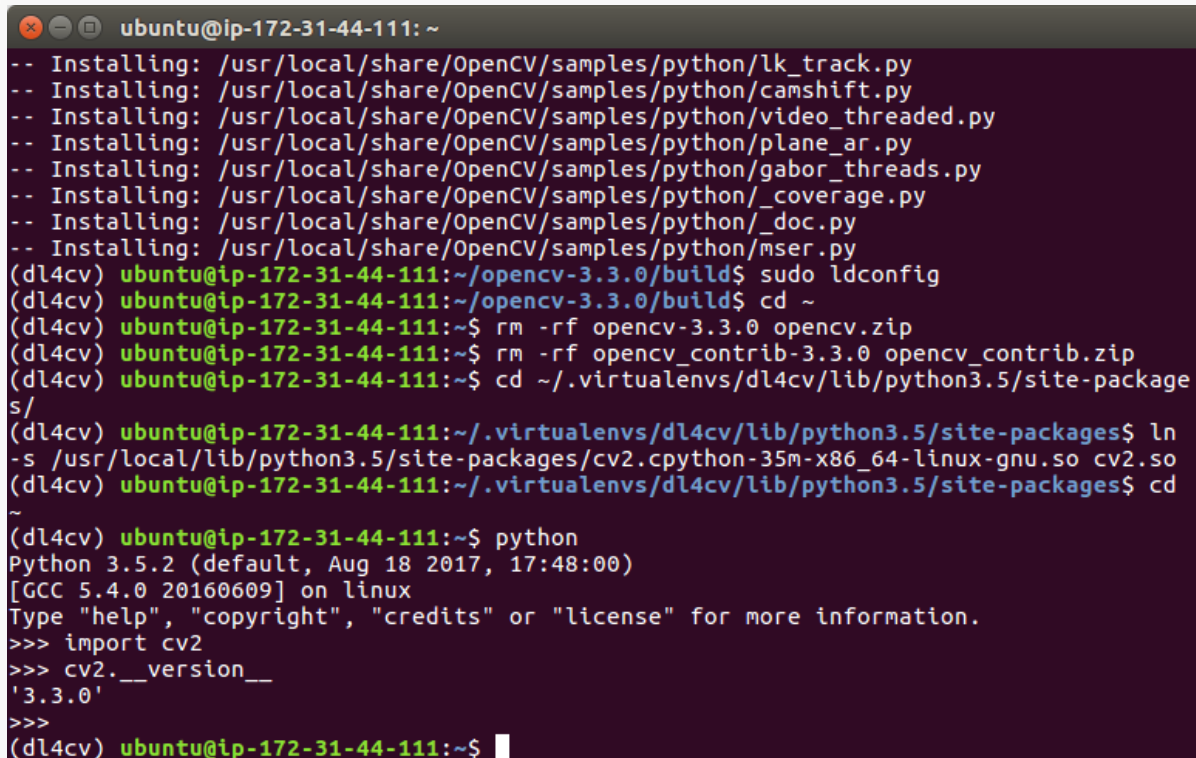
```
Setting up Ubuntu 16.04 + CUDA + GPU for deep learning with Python 3.5
1 $ python
```

```
2 >>> import cv2
3 >>> cv2.__version__
4 '3.3.0'
```

Make sure you are in the `dl4cv` virtual environment before firing up Python. You can accomplish this by running `workon dl4cv`.

When you print the OpenCV version in your Python shell it should match the version of OpenCV that you installed (in our case OpenCV `3.3.0`).

When your compilation is 100% complete you should see output that looks similar to the following:



```
ubuntu@ip-172-31-44-111: ~
-- Installing: /usr/local/share/OpenCV/samples/python/lk_track.py
-- Installing: /usr/local/share/OpenCV/samples/python/camshift.py
-- Installing: /usr/local/share/OpenCV/samples/python/video_threaded.py
-- Installing: /usr/local/share/OpenCV/samples/python/plane_ar.py
-- Installing: /usr/local/share/OpenCV/samples/python/gabor_threads.py
-- Installing: /usr/local/share/OpenCV/samples/python/_coverage.py
-- Installing: /usr/local/share/OpenCV/samples/python/_doc.py
-- Installing: /usr/local/share/OpenCV/samples/python/mser.py
(dl4cv) ubuntu@ip-172-31-44-111:~/opencv-3.3.0/build$ sudo ldconfig
(dl4cv) ubuntu@ip-172-31-44-111:~/opencv-3.3.0/build$ cd ~
(dl4cv) ubuntu@ip-172-31-44-111:~$ rm -rf opencv-3.3.0 opencv.zip
(dl4cv) ubuntu@ip-172-31-44-111:~$ rm -rf opencv_contrib-3.3.0 opencv_contrib.zip
(dl4cv) ubuntu@ip-172-31-44-111:~$ cd ~/.virtualenvs/dl4cv/lib/python3.5/site-package
s/
(dl4cv) ubuntu@ip-172-31-44-111:~/.virtualenvs/dl4cv/lib/python3.5/site-packages$ ln
-s /usr/local/lib/python3.5/site-packages/cv2.cpython-35m-x86_64-linux-gnu.so cv2.so
(dl4cv) ubuntu@ip-172-31-44-111:~/.virtualenvs/dl4cv/lib/python3.5/site-packages$ cd
~
(dl4cv) ubuntu@ip-172-31-44-111:~$ python
Python 3.5.2 (default, Aug 18 2017, 17:48:00)
[GCC 5.4.0 20160609] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> import cv2
>>> cv2.__version__
'3.3.0'
>>>
(dl4cv) ubuntu@ip-172-31-44-111:~$
```

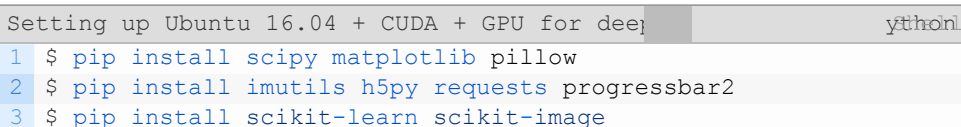
Figure 7: OpenCV 3.3.0 compilation is complete.

That's it — assuming you didn't have an import error, then you're ready to go on to **Step #6** where we will install Keras.

## Step #6: Install Keras

For this step, make sure that you are in the `dl4cv` environment by issuing the `workon dl4cv` command.

From there we can install some required computer vision, image processing, and machine learning libraries:



```
Setting up Ubuntu 16.04 + CUDA + GPU for deep learning with Python - PyImageSearch
1 $ pip install scipy matplotlib pillow
2 $ pip install imutils h5py requests progressbar2
3 $ pip install scikit-learn scikit-image
```

Next, install Tensorflow (GPU version):



```
Setting up Ubuntu 16.04 + CUDA + GPU for deep learning with Python - PyImageSearch
```



```
1 $ pip install tensorflow-gpu
```

You can verify that TensorFlow has been installed by importing it in your Python shell:

```
Setting up Ubuntu 16.04 + CUDA + GPU for deep learning with Python
1 $ python
2 >>> import tensorflow
3 >>>
```

Now we're ready to install Keras:

```
Setting up Ubuntu 16.04 + CUDA + GPU for deep learning with Python
1 $ pip install keras
```

Again, you can verify Keras has been installed via your Python shell:

```
Setting up Ubuntu 16.04 + CUDA + GPU for deep learning with Python
1 $ python
2 >>> import keras
3 Using TensorFlow backend.
4 >>>
```

You should see that Keras has been imported with no errors **and** the TensorFlow backend is being used.

Before you move on to Step #7, take a second to familiarize yourself with the `~/.keras/keras.json` file:

```
Setting up Ubuntu 16.04 + CUDA + GPU for deep learning with Python
1 {
2     "image_data_format": "channels_last",
3     "backend": "tensorflow",
4     "epsilon": 1e-07,
5     "floatx": "float32"
6 }
```

Ensure that `image_data_format` is set to `channels_last` and `backend` is `tensorflow`.

**Congratulations!** You are now ready to begin your *Deep learning for Computer Vision with Python* journey (Starter Bundle and Practitioner Bundle readers can safely skip Step #7).

## Step #7 Install mxnet (ImageNet Bundle only)

This step is only required for readers who purchased a copy of the *ImageNet Bundle* of *Deep Learning for Computer Vision with Python*. You may also choose to use these instructions if you want to configure mxnet on your system.

Either way, let's first clone the mxnet repository and checkout branch `0.11.0`:

```
Setting up Ubuntu 16.04 + CUDA + GPU for deep learning with Python
1 $ cd ~
2 $ git clone --recursive https://github.com/apache/incubator-mxnet
```

We can then compile mxnet:

```
Setting up Ubuntu 16.04 + CUDA + GPU for deep learning with Python
1 $ cd mxnet
```

```
2 $ make -j4 USE_OPENCV=1 USE_BLAS=openblas USE_CUDA=1 USE_CUDA_PATH
```

Followed by sym-linking to our dl4cv environment.

```
Setting up Ubuntu 16.04 + CUDA + GPU for deep learning with Python
1 $ cd ~/.virtualenvs/dl4cv/lib/python3.5/site-packages/
2 $ ln -s ~/mxnet/python/mxnet mxnet
3 $ cd ~
```

Finally, you may fire up Python in your environment to test that the installation was successful:

```
Setting up Ubuntu 16.04 + CUDA + GPU for deep learning with Python
1 $ python
2 >>> import mxnet
3 >>>
```

**Note:** Do not delete the `mxnet` directory in your home folder. Not only do our Python bindings live there, but we also need the files in `~/mxnet/bin` when creating serialized image datasets.

**Cheers!** You are done and deserve a cold beer while you read *Deep Learning for Computer Vision with Python* (ImageNet bundle).

**Note:** To avoid significant cloud expenses (or power bills if your box is beneath your desk), I'd recommend that you power off your machine until you're ready to use it.

## Summary

Today we learned how to set up an Ubuntu + CUDA + GPU machine with the tools needed to be successful when training your own deep learning networks.

If you encountered any issues along the way, I highly encourage you to check that you didn't skip any steps. If you are still stuck, please leave a comment below.

I want to reiterate that you don't need a fancy, expensive GPU machine to get started on your deep learning for computer vision journey. Your CPU can handle the introductory examples in the book. To help you get started, I have provided an install tutorial [here for Ubuntu CPU users](#). If you prefer the easy, pre-configured route, my book comes with a [VirtualBox virtual machine ready to go](#).

I hope this tutorial helps you on your deep learning journey!

If you want to study deep learning in-depth, be sure to take a look at my new book, *Deep Learning for Computer Vision with Python*.

To be notified when future blog posts and tutorials are published on the PyImageSearch blog, be sure to enter your email address in the form below!

**Resource Guide (it's totally free).**



Enter your email address below to get my **free 11-page Image Search Engine Resource Guide PDF**. Uncover **exclusive techniques** that I don't publish on this blog and start building image search engines of your own!

DOWNLOAD THE GUIDE!

deep learning, dl4cv, gpu, install, optimization, python, python  
3, ubuntu, virtual environments

## 45 Responses to *Setting up Ubuntu 16.04 + CUDA + GPU for deep learning with Python*



**Arash R** September 27, 2017 at 12:06 pm #

REPLY

I think you have to install NVIDIA driver again after installing Cuda. Or is it not required with this setup? (To be clear I followed the official NVIDIA guide and I think I remember that cuda-driver didn't work or something so I didn't install that but nonetheless I had to re-install my regular nvidia driver afterwards.)



**Adrian Rosebrock** September 28, 2017 at 9:13 am #

REPLY

I always install the kernel driver first, then the CUDA toolkit. This has always worked for me in the past.



**Andrew** September 27, 2017 at 2:13 pm #

REPLY

Thank you Adrian. Great tutorial. Since I don't have the \$129k for the DGX-1 I will be doing this on my NVIDIA Jetson TX2 😊



**Adrian Rosebrock** September 28, 2017 at 9:07 am #

REPLY

The TX2 is a lot of fun, you'll be able to execute the majority of the examples in Deep Learning for Computer Vision with Python

on it.



**Samuel** September 27, 2017 at 5:12 pm #

REPLY

Hi Adrian,

I am having trouble with Step 2 when I try to run `sudo ./NVIDIA-Linux-x86_64-375.26.run`.

This is the error that I get:

ERROR: You appear to be running an X server; please exit X before installing. For further details, please see the section INSTALLING THE NVIDIA DRIVER in the README available on the Linux driver download page at <http://www.nvidia.com>.

ERROR: Installation has failed. Please see the file `/var/log/nvidia-installer.log` for details. You may find suggestions on fixing installation problems in the README available on the Linux driver download page at <http://www.nvidia.com>.

And here is the file `/var/log/nvidia-installer.log`

nvidia-installer log file `/var/log/nvidia-installer.log`  
creation time: Wed Sep 27 12:28:34 2017  
installer version: 375.26

PATH: `/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/snap/bin`

nvidia-installer command line:  
`./nvidia-installer`

Unable to load: nvidia-installer ncurses v6 user interface

Using: nvidia-installer ncurses user interface

-> Detected 8 CPUs online; setting concurrency level to 8.

-> The file `/tmp/.X0-lock` exists and appears to contain the process ID `'1016'` of a running X server.

ERROR: You appear to be running an X server; please exit X before installing. For further details, please see the section INSTALLING THE NVIDIA DRIVER in the README available on the Linux driver download page at <http://www.nvidia.com>.

ERROR: Installation has failed. Please see the file `/var/log/nvidia-installer.log` for details. You may find suggestions on fixing installation problems in the README available on the Linux driver download page at <http://www.nvidia.com>.

I tried googling the error, but no luck so far. I did a `rm /tmp/.X0-lock` and then I didn't get the "X server error", but the installation still will not complete.

Any idea how to fix this?

Thanks



**Adrian Rosebrock** September 28, 2017 at 9:06 am #

REPLY

It sounds like you may have forgotten to disable the default Nouveau driver and reboot your machine. Please see the first few commands of “Step #2” for more information.

If you have disabled the driver, try booting directly to the terminal rather than trying to login to the GUI.



**Adrian Rosebrock** September 29, 2017 at 6:30 am #

REPLY

Hi Samuel — I updated the blog post to include “*Step #0: Turn off X server/X window system*”. This will help you resolve the error you are receiving.

Please give it a look!



**Alexander Sack** September 29, 2017 at 11:06 am #

REPLY

I think most would be better off doing a “network” based install which would mean adding the official nvidia ppa repos to so they can just do:

```
apt-get install cuda
```

And that's that. Note that very recently Nvidia has released full support for cuda-9.0 on 17.04 with all the trimmings (so far works flawlessly with Keras/TF etc.).

Also, you don't \*have\* to disable X11. X11 can co-exist.

Finally, I highly recommend Anaconda for machine learning projects: <http://www.anaconda.org>

For those of you who aren't familiar with Anaconda think of it as virtualenv+pip all rolled into one. OpenCV, tensorflow-gpu, keras, scikit, etc. have all been ported over to conda. Anaconda also applies to OSX as well.

Anyway, food for thought.



**Adrian Rosebrock** October 2, 2017 at 10:15 am #

REPLY

1. You can use apt-get to install the NVIDIA drivers, but I can't recommend it. You can easily place yourself in a situation when you run an upgrade and install non-compatible kernel drivers.
2. When installing the NVIDIA drivers via the command line I needed to turn off X11. Perhaps I'm missing something?
3. Regarding Anaconda, please see my reply to "Shannon".



**Azam** September 29, 2017 at 12:52 pm #

REPLY

Thanks. I will try this to setup my machine for deep learning. I have Nvidia 1060. Would be enough for deep learning?



**Adrian Rosebrock** October 2, 2017 at 10:13 am #

REPLY

Is it the 1060 6GB or 3GB model? You'll be able to run the majority of the examples inside the Starter Bundle and Practitioner Bundle of [Deep Learning for Computer Vision with Python](#) using 3GB. However, for deeper networks on larger datasets I would suggest at least 6GB, but ideally 8GB+.



**Hubert de Lassus** September 29, 2017 at 1:44 pm #

REPLY

On a fresh Ubuntu 16.04.3 installation, Nvidia driver version 375.66 is built in Ubuntu. By default the nouveau driver is installed but the Nvidia driver is available.

In graphic mode going to settings->Software & updates -> Additional Drivers select NVIDIA driver, click apply changes and reboot. This installs the Nvidia driver.

After this I was able to install Cuda 8.0 following the steps mentioned in this blog.

Attempting to install Nvidia driver as described by Adrian failed in my case even though the Xserver was disabled. So I suggest to use ubuntu settings->Software->Additional drivers to install the driver.



**Hubert de Lassus** September 29, 2017 at 2:00 pm #

REPLY

Adrian, could you please clarify:



the blog states "For this tutorial I used cuDNN v6.0 for Linux which is what TensorFlow requires."

But then the blog uses cudnn-8.0:

```
scp -i EC2KeyPair.pem ~/Downloads/cudnn-8.0-linux-x64-v6.0.tgz \  
username@your_ip_address:~
```

Is cudnn-8.0 the version you advise?



**Adrian Rosebrock** October 2, 2017 at 10:10 am #

REPLY

I saw your other comment, but I wanted to reply for other readers:

You need CUDA v8 for TensorFlow and cuDNN v6.



**Hubert de Lassus** September 29, 2017 at 2:22 pm #

REPLY

Ok my mistake, it is clear that cudnn to use is 8-0 version v6 for Tensorflow.



**john** September 29, 2017 at 5:08 pm #

REPLY

hi Adrian.

I have a question regarding ubuntu 16.04 and loading the Nvidia CUDA driver.

it seems to me that the instructions for loading the nvidia driver are for ubuntu 14, and below, which is the exact method I had to always use before to get the nvidia driver loaded and use CUDA

With Ubuntu 16.04 , all you need to do is go to Software & Updates -> Additional Drivers, and the Nvidia binary driver shows up , just load it , and thats it , no need to turn off X or go into the command line, turn off nouveau , lightdm stop/start etc . CUDA and all its examples once compiled work fine with no issue. At least thats what I did months ago and have had no issues using the GPU and CUDA. After a reboot all the latest / new Nvidia drivers as released also show up tin Additiona Drivers, to load if you want to . The driver that initially shows up is 381.22 but after that later drivers up to 384.90 show up to easily load.

Not sure if Im missing something loading the Nvidia driver with this easy method

**Adrian Rosebrock** September 30, 2017 at 9:42 am <#>

REPLY

Thanks for sharing, John! I normally use a terminal when using Ubuntu, a rarely use the GUI. I haven't tried this GUI-based method, but it seems like it can work. I just want to note to other readers that if they are using EC2 they'll need to use the pure terminal-based option.

**Vic Jackson** September 29, 2017 at 5:41 pm <#>

REPLY

Adrian,

Thanks for this fantastic guide.

I just have one question regarding the "Note: To avoid significant cloud expenses (or power bills if your box is beneath your desk), I'd recommend that you power off your machine until you're ready to use it."

Is there something about these modules/drivers/packages that would required a higher than normal idle power consumption?

Just curious!

**Adrian Rosebrock** September 30, 2017 at 9:40 am <#>

REPLY

GPUs can require considerable more energy than your CPU. That note is really for readers using the EC2 cloud where the machine is billed on hourly usage. It's just a quick reminder to shut down your instance so you don't get charged hourly. The idle power consumption in your home/apartment is not that much, just be aware that the GPU requires more energy when it's under full load.

**Michael Alex** September 29, 2017 at 10:16 pm <#>

REPLY

Hi Adrian,

If we already have successfully configured Ubuntu for Python with a CPU and compiled OpenCV, etc., must you do the full GPU configuration from scratch? Is it possible to "add on" the GPU capability in a different virtual environment? Thanks very much.

**Adrian Rosebrock** September 30, 2017 at 9:39 am <#>

REPLY



If you have already configured your dev environment for the CPU, simply install the GPU drivers, then install tensorflow-gpu into your "dl4cv" Python virtual environment.



**jerry** September 30, 2017 at 8:00 am <#>

[REPLY](#)

Hi Adrian

The latest Nvidia Cuda is Cuda 9 not Cuda 8 and Cuda driver 384.81 should we be using Cuda 9 ?



**Adrian Rosebrock** September 30, 2017 at 9:37 am <#>

[REPLY](#)

You should be using CUDA 8. TenorFlow (currently) requires CUDA 8.



**Susie** September 30, 2017 at 4:46 pm <#>

[REPLY](#)

Hi Adrian

I been following your posts and your books for a while, thank you for the great post as always!

I purchased a gaming desktop with GTX 1080 ti today I am planning to install a second GPU and I wonder if the above processes changes with the a second GPU... I found little info on doing SLI with ubuntu 16.04 and since you have 4 GPUs on your workstation I wonder if you can provide some insights.



**Adrian Rosebrock** October 2, 2017 at 9:49 am <#>

[REPLY](#)

Nothing will change with multiple GPUs. I would suggest ensuring all your GPUs are the same model and then install CUDA + cuDNN. Run `nvidia-smi` and you should see that all your GPUs are recognized.



**vahid** October 1, 2017 at 6:32 am <#>

[REPLY](#)

thanks adrian very use full



**Alan** October 1, 2017 at 11:03 am #

REPLY

Hi Adrian,

I have an NVIDIA 730GT video card, I have checked on the NVIDIA site and it appears to have a compute value of 3.5, however before I follow this excellent GPU support document is there any chance of you reviewing the card for GPU compatibility please (a bit cheeky of me I know)



**Lavanya Seetharaman** October 1, 2017 at 10:34 pm #

REPLY

Hi Adrian ,

Excellent blog which I never seen before  
Whether I can try this same steps in virtual box?



**Adrian Rosebrock** October 2, 2017 at 9:37 am #

REPLY

VirtualBox, by definition, cannot access external peripherals such as your GPU. You would need to use NVIDIA's Docker image or configure your own native Ubuntu + GPU system.



**lightman** October 5, 2017 at 7:49 am #

REPLY

Excellent blog.

Should I install cuBLAS in Figure 2?



**Adrian Rosebrock** October 6, 2017 at 5:03 pm #

REPLY

You want to if you can, but it is not required.



**Abderrazak IAZZI** October 6, 2017 at 9:22 am #

REPLY

Hello all,

I have a problem with installation of nvidia driver. I have nvidia geforce gtx 960m for notbook computer. When i follow the above prompts especially when i use the command "\$ modprobe nvidia ". I got this error : " ERROR: could not insert 'nvidia' : required key not available " . i tried many tutorials but they are not clear as what you presented here.