Configuring Ubuntu for deep learning with Python

by **Adrian Rosebrock** on September 25, 2017 in **Deep Learning**, **DL4CV**

When it comes to learning new technology such as deep learning, configuring your development environment tends to be half the battle. Different operating systems, hardware, dependencies, and the actual libraries themselves can lead to many headaches before you're even able to get started studying deep learning.

These issues are further compounded by the speed of deep learning library updates and releases — new features push innovation, but oftentimes break previous versions. Your environment can quickly become obsolete, so it is imperative to become an expert in installing and configuring your deep learning environment.

Now that *Deep Learning for Computer Vision with Python* has officially released, I'll be publishing three posts this week where I will demonstrate how to stand up your own deep learning environments so that you can get a head start before you dive into reading.

I'll be demonstrating how to configure your own native development environment for the following operating systems and peripherals:

- Configuring Ubuntu for deep learning with Python (i.e., the post you are currently reading)
- Setting up Ubuntu (with GPU support) for deep learning with Python
- Configuring macOS for deep learning with Python

As you start to walk the path to deep learning and computer vision mastery, I'll be right there with you.

To get started configuring your Ubuntu machine for deep learning with Python, *just keep reading.*

Configuring Ubuntu for deep learning with Python

Accompanying my new deep learning book is a downloadable pre-configured Ubuntu VirtualBox virtual machine with Keras, TensorFlow, OpenCV, and other computer vision/machine learning libraries pre-installed. By far, this is the fastest way to get up and running with *Deep Learning for Computer Vision with Python*.

That being said, it is often desirable to install your environment *on the bare metal* so that you can take advantage of your physical hardware. For the GPU install tutorial part of this series it is a *requirement* that you be on the metal — a VM just won't cut it since it doesn't have access to your physical GPU.

Today, our blog post is broken down into four relatively easy steps:

1. Step #1: Install Ubuntu system dependencies
2. Step #2: Create your Python 3 virtual environment
3. Step #3: Compile and Install OpenCV
4. Step #4: Install Keras

Taking note of the steps, you will see that *Deep Learning for Computer Vision with Python* supports **Python 3**.

Python 3 will be the standard on PyImageSearch going forward as it is stable and quite frankly the future. Many organizations have been hesitant to adopt Python 3 at first (me included, as there was no Python 3 support for OpenCV until OpenCV 3), but at this point if you don't adopt Python 3 you will be left in the dust. Expect PyImageSearch Gurus course contentto be compatible with Python 3 in the near future as well.

Notice that we have chosen Keras as our deep learning library. Keras "stands out from the rest" of the available libraries for it's ease of use and compatibility with both Tensorflow and Theano.

My deep learning book focuses on fundamentals and breaking into the field with ease rather than introducing you to a bunch of libraries — so for the *Starter Bundle* and *Practitioner Bundle*, I demonstrate various tasks and exercises with Keras (as well as implementing some basic neural network concepts by hand). The *ImageNet Bundle* takes advantage of mxnet as well.

While we will be primarily using Keras in my book, there are many deep learning libraries for Python, and I encourage you to become familiar with my top 9 favorite Python deep learning libraries.

To get started, you'll want to have some time on your hands and access to an Ubuntu machine's terminal — SSH is perfectly suitable if your box is in the cloud or elsewhere. Let's begin!

Step #1: Install Ubuntu system dependencies

The purpose of this step is to prepare your system with the dependencies necessary for OpenCV.

All steps in this tutorial will be accomplished by using your terminal. To start, open up your command line and update the `apt-get` package manager to refresh and upgrade and pre-installed packages/libraries:

```
Configuring Ubuntu for deep learning with Pyt                Shell
$ sudo apt-get update
$ sudo apt-get upgrade
```

We'll also need to install some developer tools as well as prerequisites required for image and video I/O, optimizations, and creating plots/visualizations:

```
Configuring Ubuntu for deep learning with Pyt                Shell
$ sudo apt-get install build-essential cmake git unzip pkg-config
$ sudo apt-get install libjpeg-dev libtiff5-dev libjasper-dev lib
$ sudo apt-get install libavcodec-dev libavformat-dev libswscale-
$ sudo apt-get install libxvidcore-dev libx264-dev
$ sudo apt-get install libgtk-3-dev
$ sudo apt-get install libhdf5-serial-dev graphviz
$ sudo apt-get install libopenblas-dev libatlas-base-dev gfortran
$ sudo apt-get install python-tk python3-tk python-imaging-tk
```

We'll wrap up Step #1 by installing the Python development headers and libraries for both Python 2.7 and Python 3.5 (that way you have both).

```
Configuring Ubuntu for deep learning with Pyt                Shell
$ sudo apt-get install python2.7-dev python3-dev
```

Note: *If you do not install the Python development headers and static library, you'll run into issues during* ***Step #3*** *where we run* `cmake` *to configure our build. If these headers are not installed, then the* `cmake` *command will be unable to automatically determine the proper values of the Python interpreter and Python libraries. In short, the output of this section will look "empty" and you will not be able to build the Python bindings. When you get to* ***Step #3****, take the time to compare your output of the command to mine.*

Let's continue on by creating a virtual environment to house OpenCV and Keras.

Step #2: Create your Python virtual environment

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

Installing pip

We are now ready to start configuring our Python development environment for the build. The first step is to install `pip` , a Python package manager:

```
Configuring Ubuntu for deep learning with Pyt                Shell
$ wget https://bootstrap.pypa.io/get-pip.py
$ sudo python get-pip.py
$ sudo python3 get-pip.py
```

Installing virtualenv and virtualenvwrapper

I've mentioned this in *every single install tutorial I've ever done*, but I'll say it again here today: I'm a *huge fan* of both virtualenv and virtualenvwrapper. **These Python packages allow you to create *separate, independent* Python environments for *each*project that you are working on.**

In short, using these packages allows you to solve the *"Project X depends on version 1.x, but Project Y needs 4.x* dilemma. A fantastic side effect of using Python virtual environments is that you can keep your system Python neat, tidy, and free from clutter.

While you can certainly install OpenCV with Python bindings without Python virtual environments, ***I highly recommend you use them*** as other PyImageSearch tutorials leverage Python virtual environments. I'll also be assuming that you have both `virtualenv` and `virtualenvwrapper` installed throughout the remainder of this guide.

If you would like a full, detailed explanation on why Python virtual environments are a *best practice*, you should absolutely give this excellent blog post on RealPython a read. I also provide some commentary on why I personally prefer Python virtual environments in the first half of this tutorial.

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:

Configuring Ubuntu for deep learning with Pyt Shell
```
$ sudo pip install virtualenv virtualenvwrapper
$ 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:

Configuring Ubuntu for deep learning with Pyt Shell
```
# virtualenv and virtualenvwrapper
export WORKON_HOME=$HOME/.virtualenvs
export VIRTUALENVWRAPPER_PYTHON=/usr/bin/python3
source /usr/local/bin/virtualenvwrapper.sh
```

The `~/.bashrc` file is simply a shell script that Bash runs whenever you launch a new terminal. You normally use this file to set various configurations. In this case, we are setting an environment variable called `WORKON_HOME` to point to the directory where our Python virtual environments live. We then load any necessary configurations from `virtualenvwrapper` .

To update your `~/.bashrc` file simply use a standard text editor. I would recommend using `nano` , `vim` , or `emacs` . You can also use graphical editors as well, but if you're just getting started, `nano` is likely the easiest to operate.

A more simple solution is to use the `cat` command and avoid editors entirely:

Configuring Ubuntu for deep learning with Pyt Shell
```
$ echo -e "\n# virtualenv and virtualenvwrapper" >> ~/.bashrc
$ echo "export WORKON_HOME=$HOME/.virtualenvs" >> ~/.bashrc
$ echo "export VIRTUALENVWRAPPER_PYTHON=/usr/bin/python3" >> ~/.b
$ echo "source /usr/local/bin/virtualenvwrapper.sh" >> ~/.bashrc
```

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

Configuring Ubuntu for deep learning with Pyt Shell
```
$ source ~/.bashrc
```

***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).*

Creating a virtual environment for deep learning and computer vision

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.

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 new deep learning book we will use Python 3**.

With that said, for the following command, ensure your Python (`-p`) flag is set to `python3` :

```
Configuring Ubuntu for deep learning with Pyt                Shell
$ mkvirtualenv dl4cv -p python3
```

Regardless of which Python version you decide to use, the end result is that we have created a Python virtual environment named `dl4cv` (short for "deep learning for computer vision").

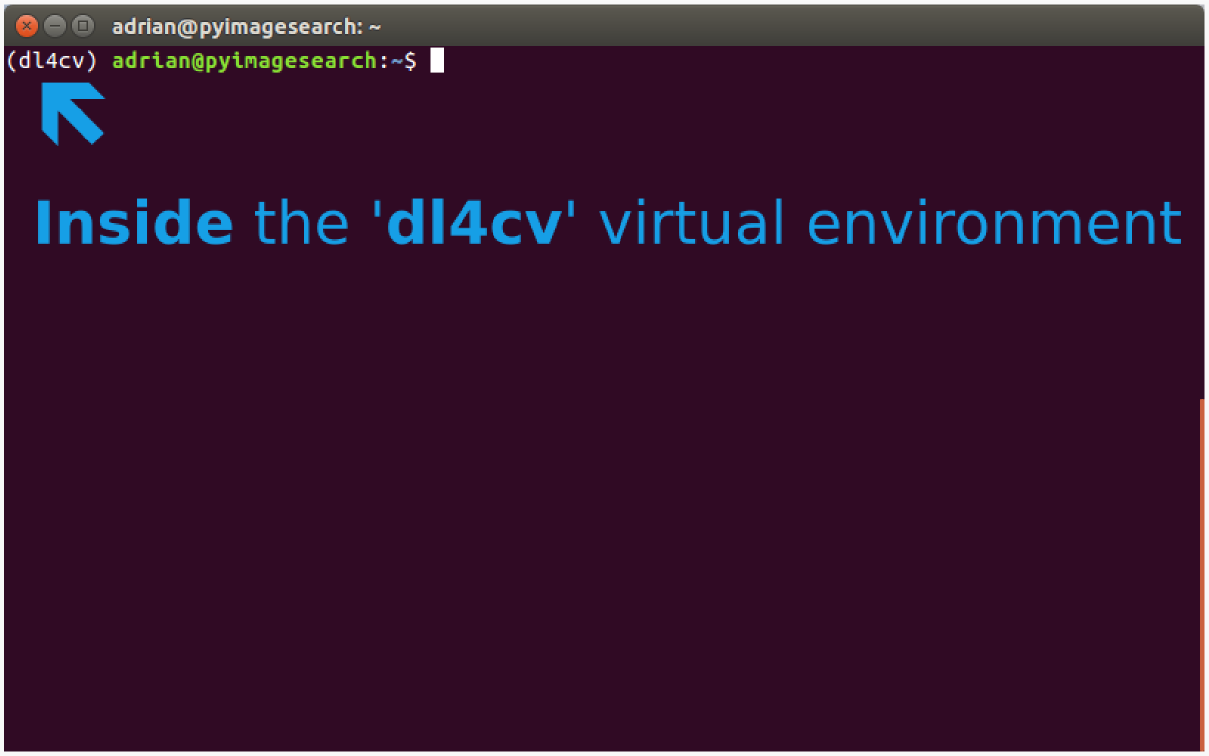
You can name this virtual environment whatever you like (and create as many Python virtual environments as you want), but for the time being, I would suggest sticking with the `dl4cv` name as that is what I'll be using throughout the rest of this tutorial as well as the remaining install guides in this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

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:

```
Configuring Ubuntu for deep learning with Pyt                Shell
$ workon dl4cv
```

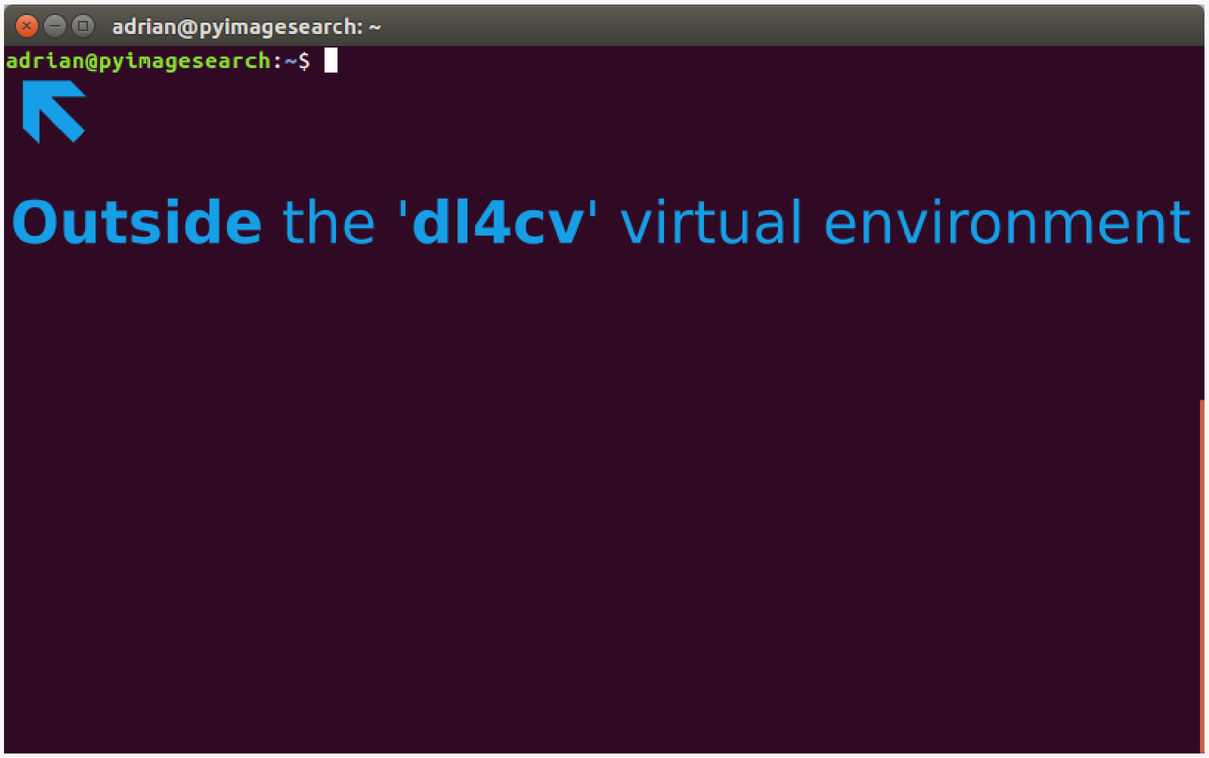
To validate that you are in the `dl4cv` virtual environment, simply examine your command line — *if you see the text `(dl4cv)` preceding your prompt, then you **are** in the `dl4cv` virtual environment:*



```
(dl4cv) adrian@pyimagesearch:~$
```

Figure 1: Inside the *dl4cv* virtual environment denoted by *'(dl4cv)'* in the prompt.

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



```
adrian@pyimagesearch:~$
```

Figure 2: Outside of the *dl4cv* virtual environment. Simply execute the *'workon dl4cv'* command to get into 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:

```
Configuring Ubuntu for deep learning with Py                Shell
$ pip install numpy
```

Step #3: Compile and Install OpenCV

In this section we will install and compile OpenCV. We'll start by downloading and unarchiving OpenCV 3.3. Then we will build and compile OpenCV from source. Finally we will test that OpenCV has been installed.

Downloading OpenCV

First let's download opencv and opencv_contrib into your home directory:

```
Configuring Ubuntu for deep learning with                   Shell
$ cd ~
$ wget -O opencv.zip https://github.com/Itseez/opencv/archive/3.3
$ wget -O opencv_contrib.zip https://github.com/Itseez/opencv_con
```

You may need to expand the commands above to copy and past the *full path* to the `opencv_contrib` file.

Then, let's unzip both files:

```
Configuring Ubuntu for deep learning with Py                Shell
$ unzip opencv.zip
$ unzip opencv_contrib.zip
```

Running CMake

Let's create a `build` directory and run CMake:

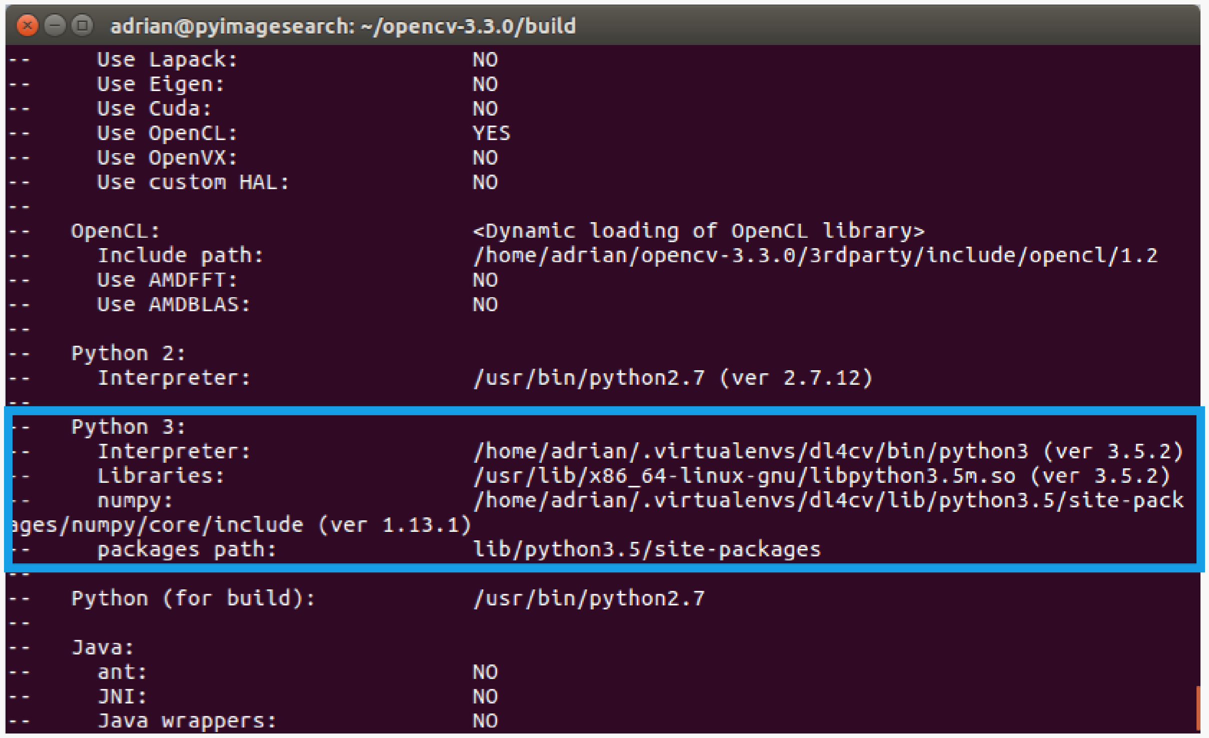
```
Configuring Ubuntu for deep learning with Py                Shell
$ cd ~/opencv-3.3.0/
$ mkdir build
$ cd build
$ cmake -D CMAKE_BUILD_TYPE=RELEASE \
    -D CMAKE_INSTALL_PREFIX=/usr/local \
    -D WITH_CUDA=OFF \
    -D INSTALL_PYTHON_EXAMPLES=ON \
    -D OPENCV_EXTRA_MODULES_PATH=~/opencv_contrib-3.3.0/modules \
    -D BUILD_EXAMPLES=ON ..
```

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:



```
adrian@pyimagesearch: ~/opencv-3.3.0/build
--     Use Lapack:                   NO
--     Use Eigen:                    NO
--     Use Cuda:                     NO
--     Use OpenCL:                   YES
--     Use OpenVX:                   NO
--     Use custom HAL:               NO
--
--   OpenCL:                         <Dynamic loading of OpenCL library>
--     Include path:                 /home/adrian/opencv-3.3.0/3rdparty/include/opencl/1.2
--     Use AMDFFT:                   NO
--     Use AMDBLAS:                  NO
--
--   Python 2:
--     Interpreter:                  /usr/bin/python2.7 (ver 2.7.12)
--
--   Python 3:
--     Interpreter:                  /home/adrian/.virtualenvs/dl4cv/bin/python3 (ver 3.5.2)
--     Libraries:                    /usr/lib/x86_64-linux-gnu/libpython3.5m.so (ver 3.5.2)
--     numpy:                        /home/adrian/.virtualenvs/dl4cv/lib/python3.5/site-pack
ages/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 3: Checking that Python 3 will be used when compiling OpenCV 3 for Ubuntu.

Pay attention 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 (I would also suggest deleting the `build` directory and re-creating it).

Compiling OpenCV

Now we are now ready to compile OpenCV with 4 cores:

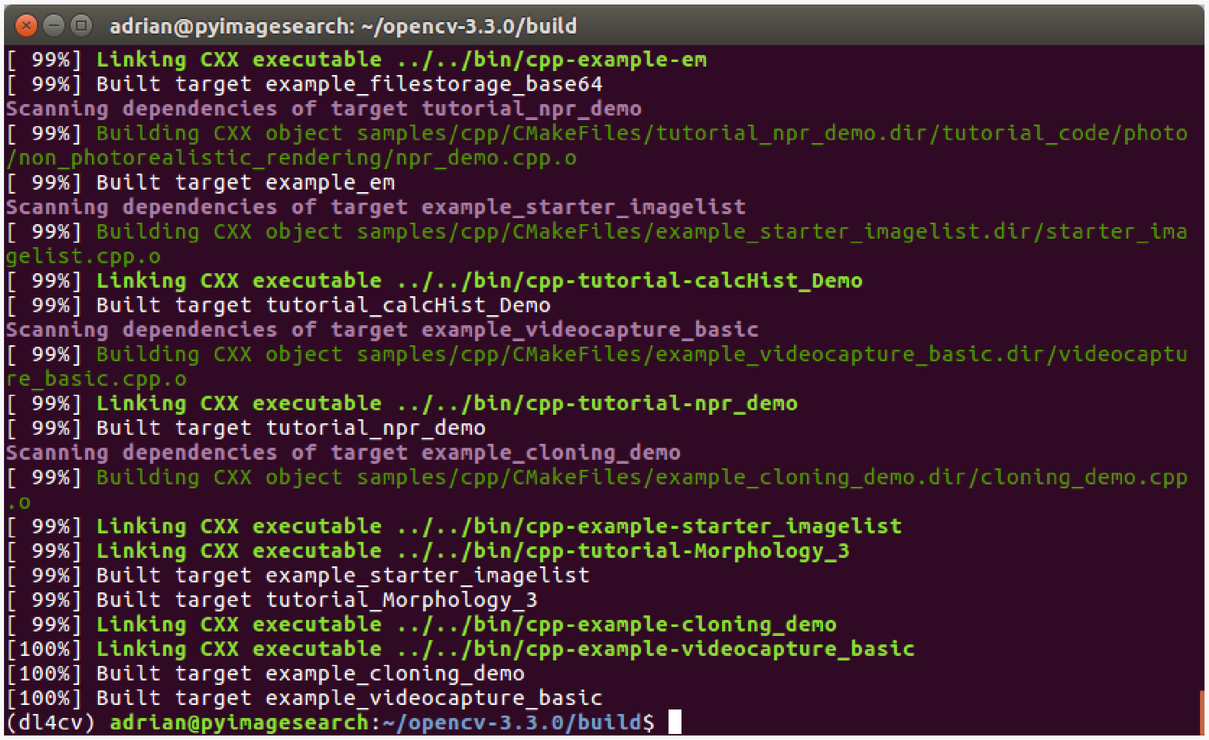
```
Configuring Ubuntu for deep learning with Pyt                    Shell
$ make -j4
```

Note: *You can try a version of the* `-j4` *flag corresponding to the number of cores of your CPU to achieve compile time speedups. In this case I used* `-j4` *since my machine has four cores. If you run into compilation errors, you may run the command* `make clean` *and then just compile without the parallel flag:* `make` .

From there, all you need to do is to install OpenCV 3.3 and then free up some disk space if you so desire:

```
Configuring Ubuntu for deep learning with Pyt                    Shell
$ sudo make install
$ sudo ldconfig
$ cd ~
$ rm -rf opencv-3.3.0 opencv.zip
$ rm -rf opencv_contrib-3.3.0 opencv_contrib.zip
```

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



```
adrian@pyimagesearch: ~/opencv-3.3.0/build
[ 99%] Linking CXX executable ../../bin/cpp-example-em
[ 99%] Built target example_filestorage_base64
Scanning dependencies of target tutorial_npr_demo
[ 99%] Building CXX object samples/cpp/CMakeFiles/tutorial_npr_demo.dir/tutorial_code/photo
/non_photorealistic_rendering/npr_demo.cpp.o
[ 99%] Built target example_em
Scanning dependencies of target example_starter_imagelist
[ 99%] Building CXX object samples/cpp/CMakeFiles/example_starter_imagelist.dir/starter_ima
gelist.cpp.o
[ 99%] Linking CXX executable ../../bin/cpp-tutorial-calcHist_Demo
[ 99%] Built target tutorial_calcHist_Demo
Scanning dependencies of target example_videocapture_basic
[ 99%] Building CXX object samples/cpp/CMakeFiles/example_videocapture_basic.dir/videocaptu
re_basic.cpp.o
[ 99%] Linking CXX executable ../../bin/cpp-tutorial-npr_demo
[ 99%] Built target tutorial_npr_demo
Scanning dependencies of target example_cloning_demo
[ 99%] Building CXX object samples/cpp/CMakeFiles/example_cloning_demo.dir/cloning_demo.cpp
.o
[ 99%] Linking CXX executable ../../bin/cpp-example-starter_imagelist
[ 99%] Linking CXX executable ../../bin/cpp-tutorial-Morphology_3
[ 99%] Built target example_starter_imagelist
[ 99%] Built target tutorial_Morphology_3
[ 99%] Linking CXX executable ../../bin/cpp-example-cloning_demo
[100%] Linking CXX executable ../../bin/cpp-example-videocapture_basic
[100%] Built target example_cloning_demo
[100%] Built target example_videocapture_basic
(dl4cv) adrian@pyimagesearch:~/opencv-3.3.0/build$
```

Figure 4: OpenCV compilation is complete.

Symbolic linking OpenCV to your virtual environment

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

```
Configuring Ubuntu for deep learning with                  Shell
$ cd ~/.virtualenvs/dl4cv/lib/python3.5/site-packages/
$ ln -s /usr/local/lib/python3.5/site-packages/cv2.cpython-35m-x8
$ cd ~
```

Notice that I am using *Python 3.5* in this example. If you are using *Python 3.6* (or newer) you'll want to update the paths to use your version of Python.

Secondly, your `.so` file (i.e., the actual OpenCV bindings) may be some variant of what is shown above, so be sure to use the appropriate file by double-checking the path.

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:

```
Configuring Ubuntu for deep learning with Py               Python
$ python
>>> import cv2
>>> cv2.__version__
'3.3.0'
```

Make sure you are in the `dl4cv` virtual environment before firing up Python (`workon dl4cv`). When you print out the version, it should match the version of OpenCV that you installed (in our case, OpenCV `3.3.0`).

That's it — assuming you didn't encounter an import error, you're ready to go on to **Step #4** where we will install Keras.

Step #4: Install Keras

For this step, be sure that you are in the `dl4cv` environment by issuing the `workon dl4cv` command. Then install our various Python computer vision, image processing, and machine learning libraries:

```
Configuring Ubuntu for deep learning with Py                Shell
$ pip install scipy matplotlib pillow
$ pip install imutils h5py requests progressbar2
$ pip install scikit-learn scikit-image
```

Next, install Tensorflow:

```
Configuring Ubuntu for deep learning with Python
$ pip install tensorflow
```

Notice how we are using the ***CPU version*** of TensorFlow. I will be covering the ***GPU version*** in a separate tutorial.

Installing Keras is extremely simple, thanks to `pip` :

```
Configuring Ubuntu for deep learning with Py                Shell
$ pip install keras
```

Again, do this in the `dl4cv` virtual environment.

You can test our Keras install from a Python shell:

```
Configuring Ubuntu for deep learning with Py               Python
$ python
>>> import keras
Using TensorFlow backend.
>>>
```

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

Before you wrap up the install tutorial take a second to familiarize yourself with the `~/.keras/keras.json` file:

```
Configuring Ubuntu for deep learning with Python
{
    "image_data_format": "channels_last",
    "backend": "tensorflow",
    "epsilon": 1e-07,
    "floatx": "float32"
}
```

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.

Summary

In today's blog post, I demonstrated how to set up your deep learning environment on an Ubuntu machine using only the CPU. Configuring your development environment is half the battle when it comes to learning new techniques, algorithms, and libraries.

If you're interested in studying deep learning in more detail, be sure to take a look at my new book, *Deep Learning for Computer Vision with Python*.

The next few blog posts in this series will cover alternative environments including macOS and Ubuntu (with GPU support).

Of course, if you're interested in *pre-configured* deep learning development environments, take a look my Ubuntu virtual machine and Amazon EC2 instance.

If you are interested in learning more about computer vision and deep learning, ***be sure to enter your email address in the form below to be notified when new blog posts + tutorials are published!***

deep learning, **dl4cv**, **install**, **python**, **python 3**, **ubuntu**, **virtual environments**

32 Responses to *Configuring Ubuntu for deep learning with Python*

Brian Robbins September 25, 2017 at 11:32 am # REPLY

libjasper-dev libpng12-dev unavailable on Ubuntu 17.04 (see https://github.com/opencv/opencv/issues/8622)
So are we forced to use Ubuntu 16?

Adrian Rosebrock September 26, 2017 at 8:23 am # REPLY

I created this tutorial with the intention of Ubuntu 16 being used. I would suggest using Ubuntu 16.

Michael Schwab September 25, 2017 at 12:38 pm # REPLY

It's also possible to use a pre-created dockerfile to get setup with a full Cuda/CuDnn/Keras/Theano/TensorFlow/jupyter/OpenCv environment in ubuntu that has access to the cpu and gpu. This is using something called nvidia-docker and is probably the easiest way to have a full running gpu stack. I've used it successfully all weekend to work through the first 12 chapters of the deeplearning book. Here's more info.. .https://github.com/floydhub/dl-docker. For me, having a full docker stack that doesn't require any changes to the host system makes things extremely portable and easy to configure.

Adrian Rosebrock September 26, 2017 at 8:21 am # REPLY

I've used Docker + GPU before, the biggest problem is that it's a huge support burden for me personally. Some PyImageSearch readers are just getting started and Docker is overkill so a simple VM with a nice GUI is more appropriate. Some readers are more advanced and want to work with the "bare metal". Docker is excellent and makes life easier and I do hope I can cover it in the future. But for the time being I simply need to create a baseline of install instructions that "work".

Jose Antonio Sanchez Valero September 25, 2017 at 1:38 pm # REPLY

How much free space we need for this kind of task? Btw I'm thinking about a new partition in my disc for deep learning with ubuntu so it would be nice if you could estimate how much we need

Regards.

Adrian Rosebrock September 26, 2017 at 8:18 am # REPLY

The more the better. I would recommend at least 16GB to start (including OS files), especially if you intend on following next week's blog post on GPU + CUDA. Keep in mind that you'll not only need space for the drivers but for any code + datasets you want to work with as well.

ofey September 25, 2017 at 1:48 pm # REPLY

Hey Adrian! Thank you for the tutorial 🙂

Can you also post a tutorial for installing CUDA support for OpenCV with Python on Ubuntu. Like I'm able to set it up but have to go through a bunch of tutorials. So if you can enumerate the steps (in your style) it would be awesome!

Adrian Rosebrock September 26, 2017 at 8:17 am # REPLY

Yes, this coming Wednesday (September 27th, 2017) I'll be doing a GPU + CUDA install tutorial for Ubuntu.

Jason September 25, 2017 at 7:38 pm # REPLY

Great little tutorial…. I'm just wondering you prefer to compile opencv from source instead of plain install? Do you get more stuff this way ??

Also what about anaconda for a virtual environment. I've been using it for quite some time without issues? Again… with anaconda I can just install opencv?!?

Adrian Rosebrock September 26, 2017 at 8:11 am # REPLY

I'm not sure what you mean by "plain install". Do you mean using `apt-get`? The reason is because the `apt-get` will install an older, out of date version of OpenCV. Compiling from source will allow you to pull in the latest OpenCV version, including the "contrib" modules. Anaconda does a better job of keeping their scientific libraries up to date, but again, compiling from source gives you the most control.

Michael September 25, 2017 at 9:32 pm # REPLY

Ugh, I got all the way to the end of Step 3, and couldn't import cv2:
Import error: No module named 'cv2'

Any suggestions?

Adrian Rosebrock September 26, 2017 at 8:10 am # REPLY

There are a number of things that could be causing the problem. First, make sure that CMake correctly configured the Python 3 build and "python3" is in the list of OpenCV modules to be compiled. Secondly, make sure you sym-linked the resulting `.so` file (which are your actual Python bindings) into the Python virtual environment. Finally, double-check that you are in the `dl4cv` Python virtual environment before running CMake.

Daniel Watanabe September 26, 2017 at 1:53 pm # REPLY

To fix the import cv2 problem i refered to this link: https://stackoverflow.com/a/34628823/7690982

Adrian Rosebrock September 27, 2017 at 6:50 am # REPLY

Are you using Python virtual environments? If so, I *do not* recommend this solution. Let your Python virtual environment set the `PYTHONPATH` accordingly.

Daniel Watanabe September 27, 2017 at 8:09 am # REPLY

Yes, I am using Python virtual environments. However, the Python wasn't setting the PythonPath.

Adrian Rosebrock September 27, 2017 at 8:40 am#

I'd be curious to learn more about your setup. Explicitly setting the PYTHONPATH shouldn't be necessary. Definitely ensure that the `.so` file (i.e., your actual OpenCV bindings) are correctly sym-linked into your Python virtual environment.

jl September 26, 2017 at 6:59 am # REPLY

i sent a message yesterday that I was having issues configuring VM , Cancel it , its all working now , I made a new virtual machine and now its all working

thanks

Adrian Rosebrock September 26, 2017 at 8:05 am # REPLY

Congrats on resolving the issue!

mario September 26, 2017 at 1:47 pm # REPLY

Hi.

This comes at a fortunate time as I was just looking for something simliar, but I'm still looking into different frameworks: what do you think about pytorch? I have no experience but the autograd feature looks like a sweet component to have especially when playing around with the architecture, do you think keras (with any backend) is flexible enough for research?

Thank you!

Adrian Rosebrock September 27, 2017 at 6:48 am # REPLY

I plan on covering PyTorch vs. Keras in a future blog post, but my opinion is that Keras is more mature, more used, and has a larger community. PyTorch has a lot of nice features, but Keras is fantastic for both research and commercial applications.

Daniel Watanabe September 26, 2017 at 1:57 pm # REPLY

For this error: /opencv_contrib-3.1.0/modules/hdf/include/opencv2/hdf/hdf5.hpp:40:18: fatal error: hdf5.h: No such file or directory
compilation terminated. I refered to the links below for help:

https://github.com/opencv/opencv/issues/6016
https://github.com/BVLC/caffe/issues/4333