

# Install-OpenCV-for Python

<https://www.pyimagesearch.com/2016/10/24/ubuntu-16-04-how-to-install-opencv/>

## 1. Install OpenCV dependencies

### 1.1. Upgrade

```
$ sudo apt-get update  
$ sudo apt-get upgrade
```

### 1.2. Install some developer tools

The **pkg-config** package is (very likely) already installed on your system, but be sure to include it in the above **apt-get** command just in case. The **cmake** program is used to automatically configure our OpenCV build.

```
$ sudo apt-get install build-essential cmake pkg-config
```

OpenCV is an image processing and computer vision library. Therefore, OpenCV needs to be able to load various image file formats from disk such as JPEG, PNG, TIFF, etc. In order to load these images from disk, OpenCV actually calls other image I/O libraries that actually facilitate the loading and decoding process. We install the necessary ones

```
$ sudo apt-get install libjpeg8-dev libtiff5-dev libjasper-dev libpng12-dev
```

Use the following commands to install packages used to process video streams and access frames from cameras

```
$ sudo apt-get install libavcodec-dev libavformat-dev libswscale-dev libv4l-dev  
$ sudo apt-get install libxvidcore-dev libx264-dev
```

OpenCV ships out-of-the-box with a very limited set of GUI tools. These GUI tools allow us to display an image to our screen ( `cv2.imshow` ), wait for/record keypresses ( `cv2.waitKey` ), track mouse events ( `cv2.setMouseCallback` ), and create simple GUI elements such as sliders and trackbars. Again, you shouldn't expect to be building full-fledged GUI applications with OpenCV — these are just simple tools that allow you to debug your code and build very simple applications.

Internally, the name of the module that handles OpenCV GUI operations is **highgui** . The **highgui** module relies on the GTK library, which you should install using the following command:

```
$ sudo apt-get install libgtk-3-dev
```

Next, we install libraries that are used to optimize various functionalities inside OpenCV, such as matrix operations:

```
$ sudo apt-get install libatlas-base-dev gfortran
```

We'll install the Python development headers and libraries for Python 3.5

```
$ sudo apt-get install python3.5-dev.
```

## 2. Download the OpenCV source

First find out what is the version of the latest OpenCV. Now it is the **3.3.1**. So in this document we use this number as version number

```
$ cd ~  
$ wget -O opencv.zip https://github.com/Itseez/opencv/archive/3.3.1.zip  
$ unzip opencv.zip  
  
$ wget -O opencv_contrib.zip https://github.com/Itseez/opencv_contrib/archive/3.3.1.zip  
$ unzip opencv_contrib.zip
```

I also want to mention that both your **opencv** and **opencv\_contrib** versions should be the same (in this case, **3.3.1**). If the versions numbers do not matchup, you could very easily run into compile time errors

## 3. Setup your Python environment

### 3.1. Install virtualenv and virtualenvwrapper

```
$ sudo pip install virtualenv virtualenvwrapper
```

Find where the **virtualenvwrapper.sh** was placed. In my case it is **\$HOME/.local/bin/virtualenvwrapper.sh**.

When this script is executed, the **\$HOME/.virtualenvs** folder will be created.

Inside this folder will be created the **virtualenvs**, as a folder, when you

### 3.2. Turn virtualwrapper on

Now we create a system variable for the **.virtualenvs** and we run the **virtualwrapper.sh**. This going to create the **.virtualenvs** folder for storing the virtual environments. To automate this, we put them into the **.bashrc**:

```
.bashrc  
PATH=$PATH:$HOME/.local/bin:$HOME/.virtualenvs  
  
export WORKON_HOME=$HOME/.virtualenvs  
source $HOME/.local/bin/virtualenvwrapper.sh
```

Now you have to reload the changes:

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

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

### 3.3. Create a virtual environment

We create `cv` virtual environment.

```
$ mkvirtualenv cv -p python3
```

### 3.4. Selecting / unselecting 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 `cv` virtual environment. In that case the name of the virtual environment between brackets will be shown on the beginning of the prompt

```
$ workon cv  
(cv) $
```

If you want to leave all virtual environments then you have to use the **deactivate** command.

### 3.5. Install NumPy package into the virtual environment

```
$ workon cv  
(cv) $ pip install numpy
```

## 4. Configuring and compiling OpenCV

At this point, all of our necessary prerequisites have been installed — we are now ready to compile and OpenCV.

Before we do that, double-check that you are in the `cv` virtual environment

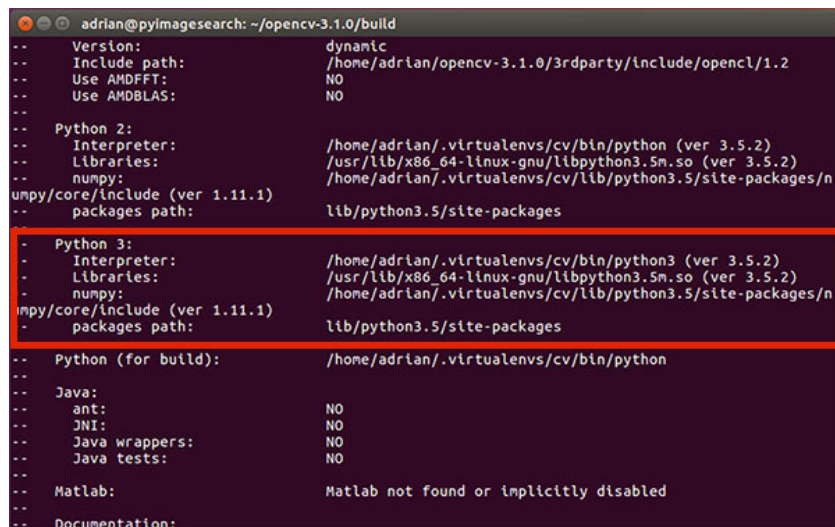
```
$ workon cv
```

### 4.1. Setup and configure build

```
$ cd /usr/local/libraries/opencv-3.3.1/  
$ mkdir build  
$ cd build  
$ cmake -D CMAKE_BUILD_TYPE=RELEASE \  
-D CMAKE_INSTALL_PREFIX=/usr/local \  
-D INSTALL_PYTHON_EXAMPLES=ON \  
-D INSTALL_C_EXAMPLES=OFF \  
-D OPENCV_EXTRA_MODULES_PATH=/usr/local/libraries/opencv_contrib-  
3.1.0/modules \  
-D PYTHON_EXECUTABLE=~/.virtualenvs/cv/bin/python \  
-D BUILD_EXAMPLES=ON ..
```

If you are getting an error related to `stdlib.h`: **No such file or directory** during either the `cmake` or `make` phase of this tutorial you'll also need to include the following option to CMake: `-D ENABLE_PRECOMPILED_HEADERS=OFF`. In this case I would suggest deleting your `build` directory, re-creating it, and then re-running CMake with the above option included. This will resolve the `stdlib.h` error.

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



```
adrian@pyimagesearch: ~/opencv-3.1.0/build  
-- Version: dynamic  
-- Include path: /home/adrian/opencv-3.1.0/3rdparty/include/opencv/1.2  
-- Use AMDFFT: NO  
-- Use AMDBLAS: NO  
--  
-- Python 2:  
-- Interpreter: /home/adrian/.virtualenvs/cv/bin/python (ver 3.5.2)  
-- Libraries: /usr/lib/x86_64-linux-gnu/libpython3.5m.so (ver 3.5.2)  
-- numpy: /home/adrian/.virtualenvs/cv/lib/python3.5/site-packages/n  
umpy/core/include (ver 1.11.1)  
-- packages path: lib/python3.5/site-packages  
--  
-- Python 3:  
-- Interpreter: /home/adrian/.virtualenvs/cv/bin/python3 (ver 3.5.2)  
-- Libraries: /usr/lib/x86_64-linux-gnu/libpython3.5m.so (ver 3.5.2)  
-- numpy: /home/adrian/.virtualenvs/cv/lib/python3.5/site-packages/n  
umpy/core/include (ver 1.11.1)  
-- packages path: lib/python3.5/site-packages  
--  
-- Python (for build): /home/adrian/.virtualenvs/cv/bin/python  
--  
-- Java:  
-- ant: NO  
-- JNI: NO  
-- Java wrappers: NO  
-- Java tests: NO  
--  
-- Matlab: Matlab not found or implicitly disabled  
--  
-- Documentation:
```

- The Interpreter points to the Python 3.5 binary in the `cv` virtual environment.
- Libraries points to the Python 3.5 library (which we installed during the final step of Step #1).
- The numpy value points to our NumPy installation in the `cv` virtual environment.

- And finally, the `packages` path points to `lib/python3.5/site-packages` . When combined with the `CMAKE_INSTALL_PREFIX` , this means that after compiling OpenCV, we'll find our `cv2.so` bindings in `/usr/local/lib/python3.5/site-packages/` .

## 4.2. Compile OpenCV

```
$ make -j4
```

The `-j` switch controls the number of processes to be used when compiling OpenCV — you'll want to set this value to the number of processors/cores on your machine. In my case, I have a quad-core processor, so I set `-j4` .

It takes time to run this command. It finishes OK when you get back the "**BUILD SUCCESSFUL**" message

Using multiple processes allows OpenCV to compile faster; however, there are times where race conditions are hit and the compile bombs out. While you can't really tell if this is the case without a lot of previous experience compiling OpenCV, if you do run into an error, my first suggestion would be to run `make clean` to flush the build, followed by compiling using only a single core:

```
$ make clean  
$ make
```

## 4.3. Install OpenCV

```
$ sudo make install  
$ sudo ldconfig
```

## 5. Finish OpenCV install

After running `sudo make install`, your OpenCV + Python 3 bindings should be located in `/usr/local/lib/python3.5/site-packages/`. Again, you can verify this using the `ls` command:

```
$ ls -l /usr/local/lib/python3.5/site-packages/  
total 1972  
-rw-r--r-- 1 root staff 2016816 Sep 13 17:24 cv2.cpython-35m-i386-linux-gnu.so
```

### 5.1. Rename the module

Again, I have no idea exactly why this happens, but it's a very easy fix. All we need to do is rename the file:

```
$ cd /usr/local/lib/python3.5/site-packages/  
$ sudo mv cv2.cpython-35m-i386-linux-gnu.so cv2.so
```

After renaming `cv2.cpython-35m-i386-linux-gnu.so` to simply `cv2.so`, we can symlink our OpenCV bindings into the `cv` virtual environment for Python 3.5:

```
$ cd ~/.virtualenvs/cv/lib/python3.5/site-packages/  
$ ln -s /usr/local/lib/python3.5/site-packages/cv2.so cv2.so
```

## 6. Testing OpenCV install

Execute the following commands to find out if your OpenCV is ready to use:

```
$ cd ~  
$ workon cv  
(cv) $ python  
>>> import cv2  
>>> cv2.__version__  
'3.3.1'
```

It works OK. Now you can delete `/usr/local/libraries/opencv-3.3.1/` and `/usr/local/libraries/opencv_contrib-3.3.1/` folders and their zip files as well.