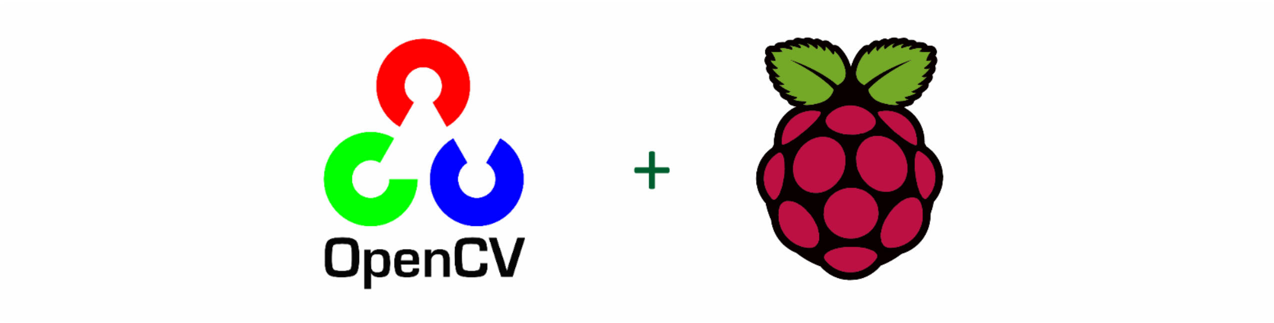


Install OpenCV 4.4.0 on Raspberry 64 OS

← 64-OS OpenCV 32-OS OpenCV →

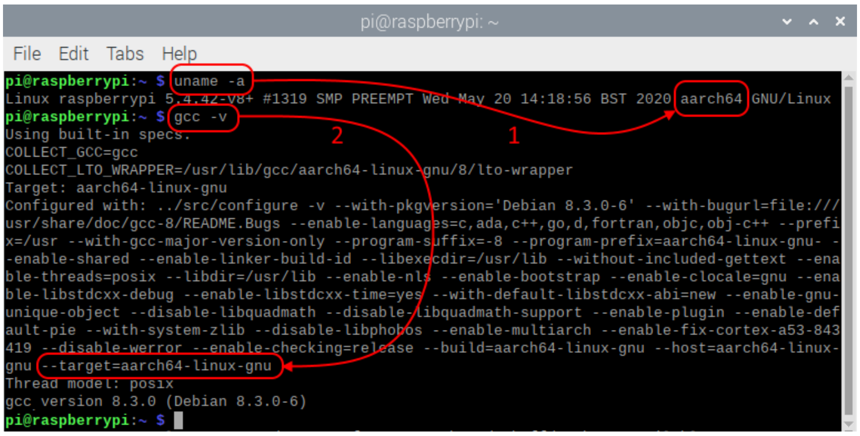
Introduction.

The Raspberry Pi is moving towards a 64-bit operating system. Within a year or so, the 32-bit OS will be fully replaced by the faster 64-bit version.

The Raspberry Foundation has recently released a more than functional beta version. Installation instructions can be found here. This guide will install OpenCV 4.4.0 on a Raspberry Pi 4 with a **64-bit** operating system.

Version check.

Before installing OpenCV 4.4.0 on your Raspberry 64-bit OS, you should first check your version. Run the command uname -a and verify your version with the screen dump below.

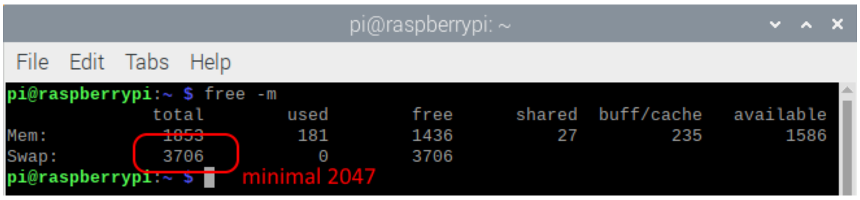


```
pi@raspberrypi:~ $ uname -a
Linux raspberrypi 5.4.42-v8+ #1319 SMP PREEMPT Wed May 20 14:18:56 BST 2020 aarch64 GNU/Linux
pi@raspberrypi:~ $ gcc -v
Using built-in specs.
COLLECT_GCC=gcc
COLLECT_LTO_WRAPPER=/usr/lib/gcc/aarch64-linux-gnu/8/lto-wrapper
Target: aarch64-linux-gnu
Configured with: ../src/configure -v --with-pkgversion='Debian 8.3.0-6' --with-bugurl=file:///
usr/share/doc/gcc-8/README.Bugs --enable-languages=c,ada,c++,go,d,fortran,objc,obj-c++ --prefi
x=/usr --with-gcc-major-version-only --program-suffix=-8 --program-prefix=aarch64-linux-gnu- -
-enable-shared --enable-linker-build-id --libexecdir=/usr/lib --without-included-gettext --ena
ble-threads=posix --libdir=/usr/lib --enable-nls --enable-bootstrap --enable-clocale=gnu --ena
ble-libstdcxx-debug --enable-libstdcxx-time=yes --with-default-libstdcxx-abi=new --enable-gnu-
unique-object --disable-libquadmath --disable-libquadmath-support --enable-plugin --enable-def
ault-pie --with-system-zlib --disable-libphobos --enable-multiarch --enable-fix-cortex-a53-843
419 --disable-werror --enable-checking=release --build=aarch64-linux-gnu --host=aarch64-linux-
gnu --target=aarch64-linux-gnu
Thread model: posix
gcc version 8.3.0 (Debian 8.3.0-6)
pi@raspberrypi:~ $
```

You also need to check your C++ compiler version with the command `gcc -v`. It must also be an `aarch64-linux-gnu` version, as shown in the screenshot. If you have a 64-bit operating system, but your gcc version is different from the one given above, reinstall the whole operating system with the latest version. The guide is found here: Install 64 bit OS on Raspberry Pi 4. You must have a 64-bit C ++ compiler as we are going to build all libraries from scratch. Otherwise, there is no point in building a 32-bit version on a 64 machine.

Swap check.

The next check is the size of the memory swap. It must be large enough to support the building. You should have at least 2 Gbyte RAM and 2 Gbyte swap space. If you have installed the 64-bit OS according to our guide, this should be no problem. Otherwise, enlarge your memory swap size.



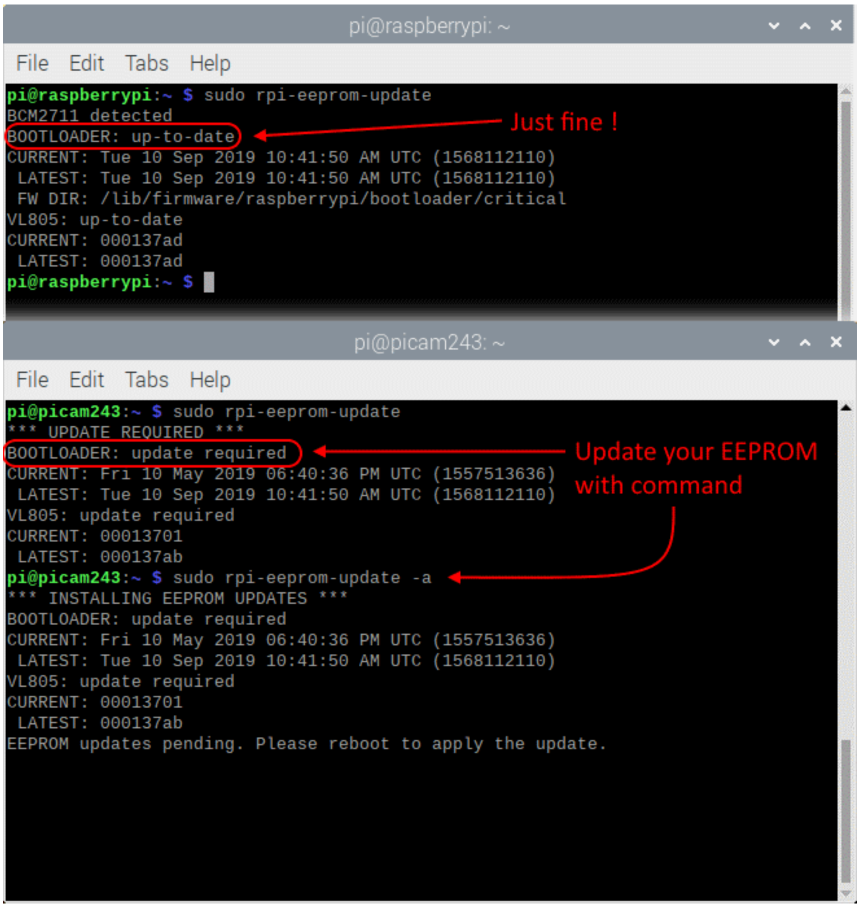
```
pi@raspberrypi:~ $ free -m
              total        used        free      shared  buff/cache   available
Mem:           1853         181        1436          27         235        1586
Swap:          3706           0        3706
pi@raspberrypi:~ $
```

EEPROM check.

The last check is the EEPROM software version. The Raspberry Pi 3 had all the operating software on the SD card. The Raspberry Pi 4, on the other hand, is also partially booted from two EEPROMs. These EEPROMs are programmed after PCB assembly in the factory. The Raspberry foundation has recently released new and improved software for these EEPROMs. This nothing to do with OpenCV, but all the more with heat dissipation. In one of our vision applications, the heat of the CPU drops from 65 °C (149 °F) to 48 °C (118 °F) simply by updating the EEPROMs contents. And, as you know, a low CPU temperature will prolong your Pi lifespan. For more information see this article.

Check, and if needed update, the EEPROMs with the following commands. The screen dumps speak for there self.

```
# to get the current status
$ sudo rpi-eeprom-update

# if needed, to update the firmware
$ sudo rpi-eeprom-update -a

$ reboot
```



```
pi@raspberrypi:~ $ sudo rpi-eeprom-update
BCM2711 detected
BOOTLOADER: up-to-date
CURRENT: Tue 10 Sep 2019 10:41:50 AM UTC (1568112110)
 LATEST: Tue 10 Sep 2019 10:41:50 AM UTC (1568112110)
 FW DIR: /lib/firmware/raspberrypi/bootloader/critical
VL805: up-to-date
CURRENT: 000137ad
 LATEST: 000137ad
pi@raspberrypi:~ $
```

```
pi@picam243:~ $ sudo rpi-eeprom-update
*** UPDATE REQUIRED ***
BOOTLOADER: update required
CURRENT: Fri 10 May 2019 06:40:36 PM UTC (1557513636)
 LATEST: Tue 10 Sep 2019 10:41:50 AM UTC (1568112110)
VL805: update required
CURRENT: 00013701
 LATEST: 000137ab
pi@picam243:~ $ sudo rpi-eeprom-update -a
*** INSTALLING EEPROM UPDATES ***
BOOTLOADER: update required
CURRENT: Fri 10 May 2019 06:40:36 PM UTC (1557513636)
 LATEST: Tue 10 Sep 2019 10:41:50 AM UTC (1568112110)
VL805: update required
CURRENT: 00013701
 LATEST: 000137ab
EEPROM updates pending. Please reboot to apply the update.
```

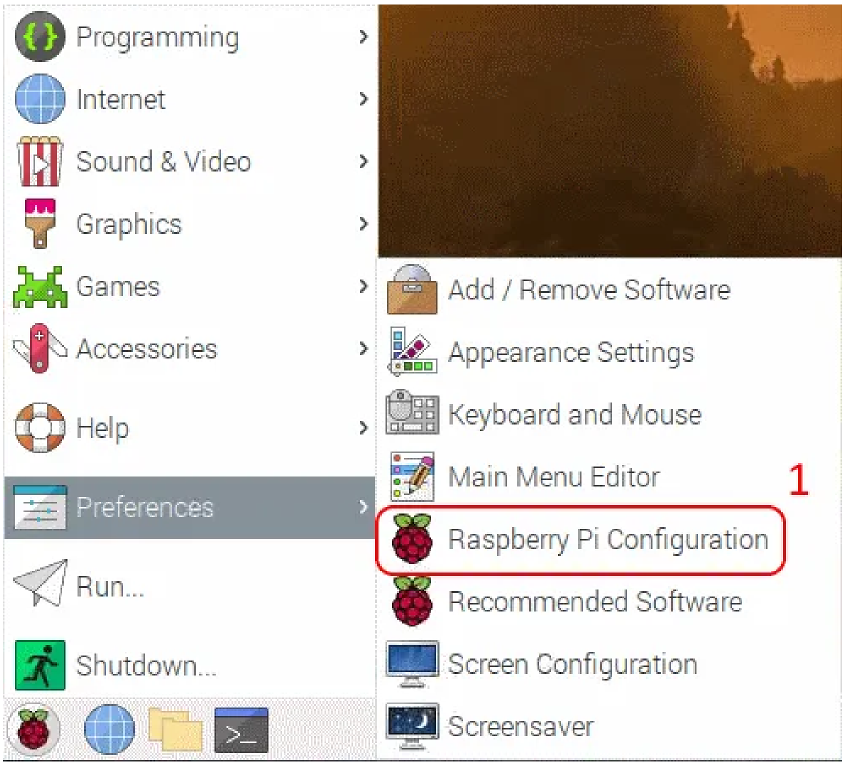
Bad ideas.

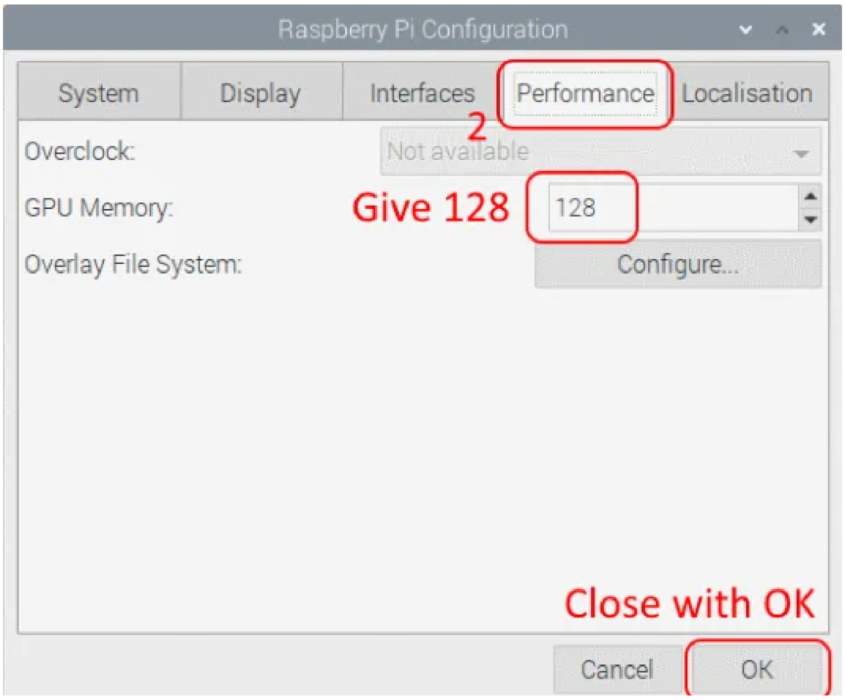
Some words of warning. Do not use `pip3` to install OpenCV on your Raspberry Pi. Nor use `sudo apt-get install`. All versions are not up to date and certainly not 64 bit. The only proper way to install OpenCV 4.4.0 is by building it from source.

GPU memory.

As explained here, the physical RAM chip is used both by the CPU and the GPU. Check and change the amount to at least

128 Mbyte if necessary, using the following menu.




Raspberry Pi Configuration

Overclock: Not available

GPU Memory: Give 128 128

Overlay File System: Configure...

Close with OK

Dependencies.

The OpenCV software uses other third party software libraries. These must be installed first. Some come with the Raspberry 64-bit operating system, others may already be installed. Better to be safe than sorry so here's the full list. Only the latest packages are installed according to the procedure.

```
# check for updates (64-bit OS is still under development!)
$ sudo apt-get update
$ sudo apt-get upgrade
# dependencies
$ sudo apt-get install build-essential cmake git unzip pkg-config
$ sudo apt-get install libjpeg-dev libpng-dev
$ sudo apt-get install libavcodec-dev libavformat-dev libswscale-dev
$ sudo apt-get install libgtk2.0-dev libcanberra-gtk* libgtk-3-dev
$ sudo apt-get install libxvidcore-dev libx264-dev

$ sudo apt-get install python3-dev python3-numpy python3-pip
$ sudo apt-get install libtbb2 libtbb-dev libdc1394-22-dev
$ sudo apt-get install libv4l-dev v4l-utils
$ sudo apt-get install libopenblas-dev libatlas-base-dev libblas-dev
$ sudo apt-get install liblapack-dev
$ sudo apt-get install liblapack-dev
gfortran libhdf5-dev

$ sudo apt-get install libprotobuf-dev
libgoogle-glog-dev libgflags-dev

$ sudo apt-get install protobuf-compiler
```

Download OpenCV.

When all third-party software is installed, OpenCV itself can be downloaded. There are two packages needed; the basic version and the additional contributions. Check before downloading the latest version at https://opencv.org/releases/. If necessary, change the names of the zip files according to the latest version. After downloading, you can unzip the files. Please be aware of line wrapping in the text boxes. The two commands are starting with `wget` and ending with `zip`.

```
$ cd ~

$ wget -O opencv.zip
https://github.com/opencv/opencv/archive/4.4.0.zip

$ wget -O opencv_contrib.zip
https://github.com/opencv/opencv_contrib/archive/4.4.0.zi


$ unzip opencv.zip

$ unzip opencv_contrib.zip
```

The next step is some administration. Rename your directories with more convenient names like `opencv` and `opencv_contrib`. This makes live later on easier.

```
$ mv opencv-4.4.0 opencv

$ mv opencv_contrib-4.4.0 opencv_contrib
```

Virtual environment.

Now it is time to decide whether or not to use a virtual environment for your OpenCV installation. We don't use the virtual environment. Instead, we simply swap the SD-card if needed. However, feel free to install a virtual environment. All instructions are given in the 32-bit OpenCV guide.

Build Make.

Before we begin with the actual build of the library, there is one small step to go. You have to make a directory where all the build files can be located.

```
$ cd ~/opencv/
$ mkdir build
$ cd build
```

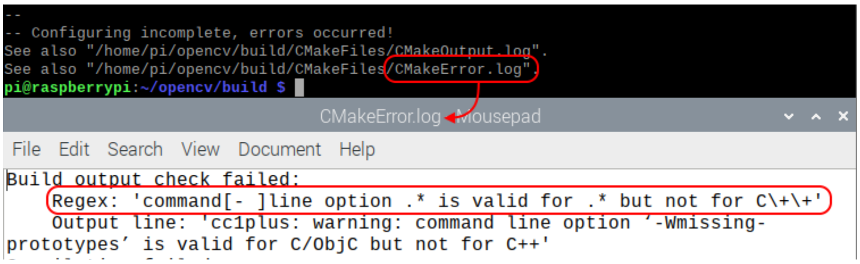
Now it is time for an important step. Here you tell CMake what, where and how to make OpenCV on your Raspberry. There are many flags involved. The most you will recognize. We save space by excluding any (Python) examples or tests.

There are only bare spaces before the -D flags, not tabs. By the way, the two last dots are no typo. It tells CMake where it can find its CMakeLists.txt (the large recipe file); one directory up.

```
$ cmake -D CMAKE_BUILD_TYPE=RELEASE \
        -D CMAKE_INSTALL_PREFIX=/usr/local \
        -D
        -D
OPENCV_EXTRA_MODULES_PATH=~/opencv_contrib/modules
\
        -D ENABLE_NEON=ON \
        -D WITH_FFMPEG=ON \
        -D WITH_GSTREAMER=ON \
        -D WITH_TBB=ON \
        -D BUILD_TBB=ON \
        -D BUILD_TESTS=OFF \
        -D WITH_EIGEN=OFF \
        -D WITH_V4L=ON \
        -D WITH_LIBV4L=ON \
        -D WITH_VTK=OFF \
        -D OPENCV_ENABLE_NONFREE=ON \
        -D INSTALL_C_EXAMPLES=OFF \
        -D INSTALL_PYTHON_EXAMPLES=OFF \
        -D BUILD_NEW_PYTHON_SUPPORT=ON \
        -D BUILD_opencv_python3=TRUE \
        -D OPENCV_GENERATE_PKGCONFIG=ON \
        -D BUILD_EXAMPLES=OFF ..
```

Please note the absence of the `ENABLE_VFPV3=ON` flag used by the building of OpenCV for the 32-bits version.

If you had used this flag, cmake would generate erros. See the screendum below.



```
--
-- Configuring incomplete, errors occurred!
See also "/home/pi/opencv/build/CMakeFiles/CMakeOutput.log".
See also "/home/pi/opencv/build/CMakeFiles/CMakeError.log".
pi@raspberrypi:~/opencv/build $
```

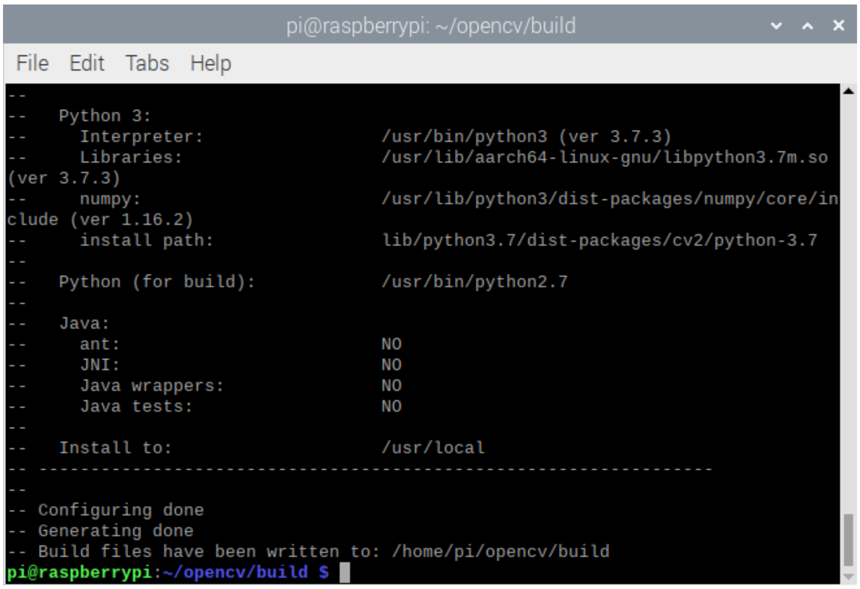
CMakeError.log - Mousepad
File Edit Search View Document Help

```
Build output check failed:
    Regex: 'command[- ]line option .* is valid for .* but not for C\+\+'
    Output line: 'cc1plus: warning: command line option ’-Wmissing-
prototypes’ is valid for C/ObjC but not for C++'
```

```
Compilation failed:
    source file: '/home/pi/opencv/build/CMakeFiles/CMakeTmp/src.cxx'
    check option: ' -fsigned-char -W -Wall -Werror=return-type -
Werror=non-virtual-dtor -Werror=address -Werror=sequence-point -Wformat -
Werror=format-security -Wmissing-declarations  -Wmissing-prototypes'
===== BUILD LOG =====
Change Dir: /home/pi/opencv/build/CMakeFiles/CMakeTmp
```

The error is quite misleading; `Regex: "command line option. * Is valid for. * But not for C ++ '`. This suggests an incompatibility of compilers instead of an unavailable option. `'Option VFPV3 not applicable'` would be a better warning. OpenCV notorious gives these types of warnings, good to keep this in mind. You can simply remove all your flags from the cmake command line and see if the build now succeeds. If so, add your flags one by one. This way, your problem will be located soon.

Hopefully, everything went well and CMake comes with a report that looks something like the screenshot below.



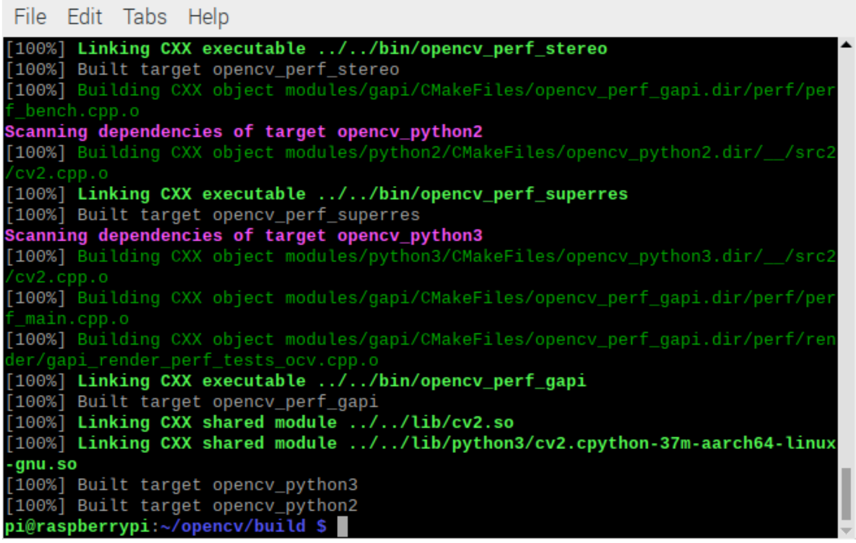
```
--
--   Python 3:
--     Interpreter:                 /usr/bin/python3 (ver 3.7.3)
--     Libraries:                   /usr/lib/aarch64-linux-gnu/libpython3.7m.so
(ver 3.7.3)
--     numpy:                       /usr/lib/python3/dist-packages/numpy/core/in
clude (ver 1.16.2)
--     install path:                lib/python3.7/dist-packages/cv2/python-3.7
--
--   Python (for build):            /usr/bin/python2.7
--
--   Java:
--     ant:                         NO
--     JNI:                         NO
--     Java wrappers:               NO
--     Java tests:                  NO
--
--   Install to:                    /usr/local
-- -----------------------------------------------------------------
--
-- Configuring done
-- Generating done
-- Build files have been written to: /home/pi/opencv/build
pi@raspberrypi:~/opencv/build $
```

Make.

With all compilation directives in place, you can start the build with the following command. This will take a while (± 50 min).

```
$ make -j4
```

Hopefully, your build was as successful as the one below.



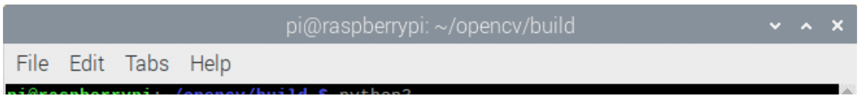
```
[100%] Linking CXX executable ../../bin/opencv_perf_stereo
[100%] Built target opencv_perf_stereo
[100%] Building CXX object modules/gapi/CMakeFiles/opencv_perf_gapi.dir/perf/per
f_bench.cpp.o
Scanning dependencies of target opencv_python2
[100%] Building CXX object modules/python2/CMakeFiles/opencv_python2.dir/__/src2
/cv2.cpp.o
[100%] Linking CXX executable ../../bin/opencv_perf_superres
[100%] Built target opencv_perf_superres
Scanning dependencies of target opencv_python3
[100%] Building CXX object modules/python3/CMakeFiles/opencv_python3.dir/__/src2
/cv2.cpp.o
[100%] Building CXX object modules/gapi/CMakeFiles/opencv_perf_gapi.dir/perf/per
f_main.cpp.o
[100%] Building CXX object modules/gapi/CMakeFiles/opencv_perf_gapi.dir/perf/ren
der/gapi_render_perf_tests_ocv.cpp.o
[100%] Linking CXX executable ../../bin/opencv_perf_gapi
[100%] Built target opencv_perf_gapi
[100%] Linking CXX shared module ../../lib/cv2.so
[100%] Linking CXX shared module ../../lib/python3/cv2.cpython-37m-aarch64-linux
-gnu.so
[100%] Built target opencv_python3
[100%] Built target opencv_python2
pi@raspberrypi:~/opencv/build $
```

Now to complete, install all the generated packages to the database of your system with the next commands.

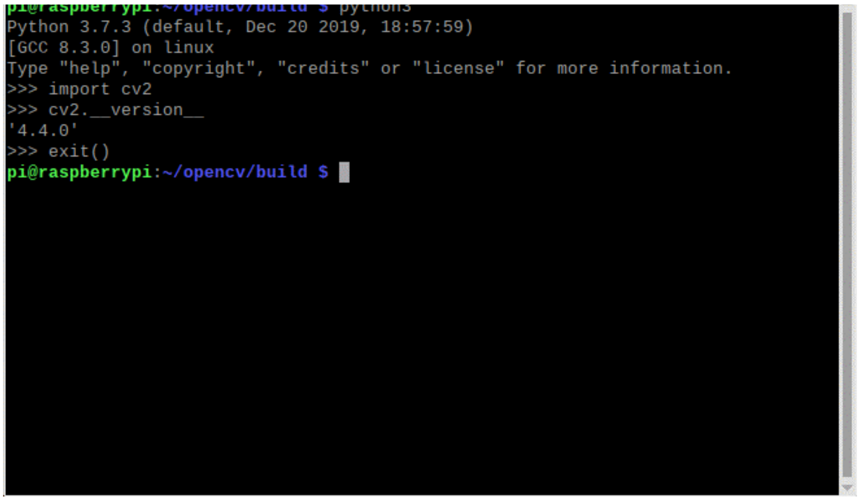
```
$ sudo make install
$ sudo ldconfig

$ sudo apt-get update
```

Checking.

Now it is time to check your installation in Python. You can use the commands as shown is the screen dump below. It all speaks for itself. Obvious, if you have installed OpenCV in a virtual environment, you need to activate this environment first with the command `workon`.



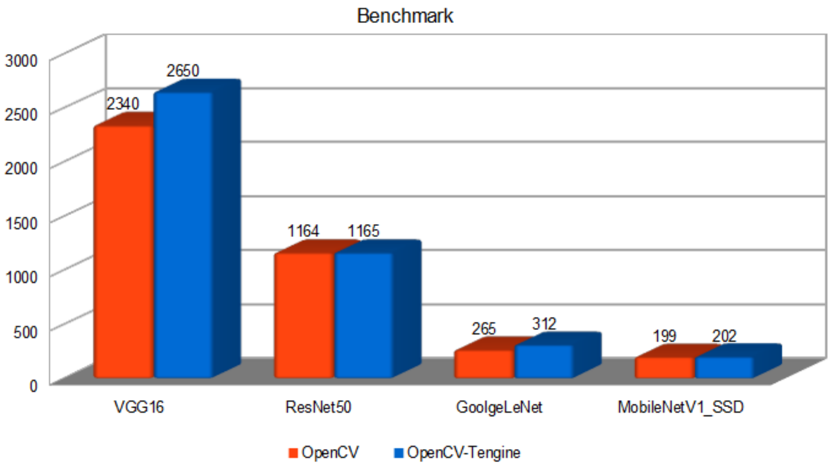
```
pi@raspberrypi:~/opencv/build $ python3
```



```
pi@raspberrypi:~/opencv/build $ python3
Python 3.7.3 (default, Dec 20 2019, 18:57:59)
[GCC 8.3.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> import cv2
>>> cv2.__version__
'4.4.0'
>>> exit()
pi@raspberrypi:~/opencv/build $
```

Tengine.

As of version 4.3.0, Tengine can be merged to the dnn (deep learning) module of OpenCV to give it an extra performance boost. There are some impressive figures in a benchmark on the GitHub page. However, after thorough testing, we failed to achieve the same results. On the contrary, the performance was reduced by the use of Tengine. The test on GitHub involved an ARM Cortex-A72 core running a single thread. Four threads at the same time, as usual with the Raspberry Pi dnn module, apparently causes problems. Below our benchmark with different models, all done with an RPi 4, 64-bit OS at 1500 MHz, timing in mSec.



Deep learning examples for →