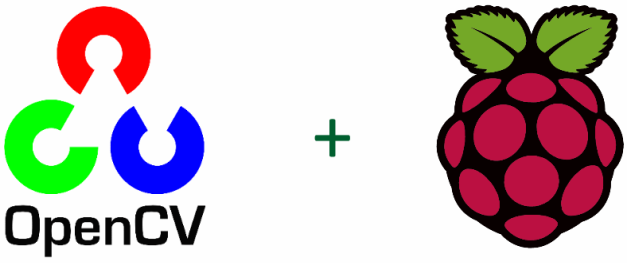


Install OpenCV 4.4.0 on Raspberry Pi 4

← OpenCV 4.3.0 64-OS OpenCV →

Introduction.

This article helps you install OpenCV 4.4.0 on Raspberry Pi 4 with a **32-bit** operation system.

Although written for the Raspberry Pi 4, the guide can also be used without any change for the Raspberry 3 or 2.

Be aware, several versions of C ++ have been released over time. All with little differences and often not fully compatible with previous versions or with the std library. Especially OpenCV, a project with many developers, is sensitive for version diversity. To keep your frustrations to a minimum, use only the most recent version of OpenCV with the most recent compilers. It becomes a whole adventure to install an old OpenCV version on a new Raspbian operating system. Many incompatible template errors have to be solved.

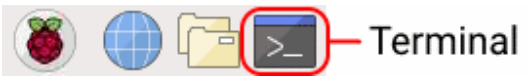
That is why Pythonians love their virtual environment: put all the software and necessary libraries in a separate directory and keep this locked for the future. This technique can result in many identical copies, that is the price to pay.

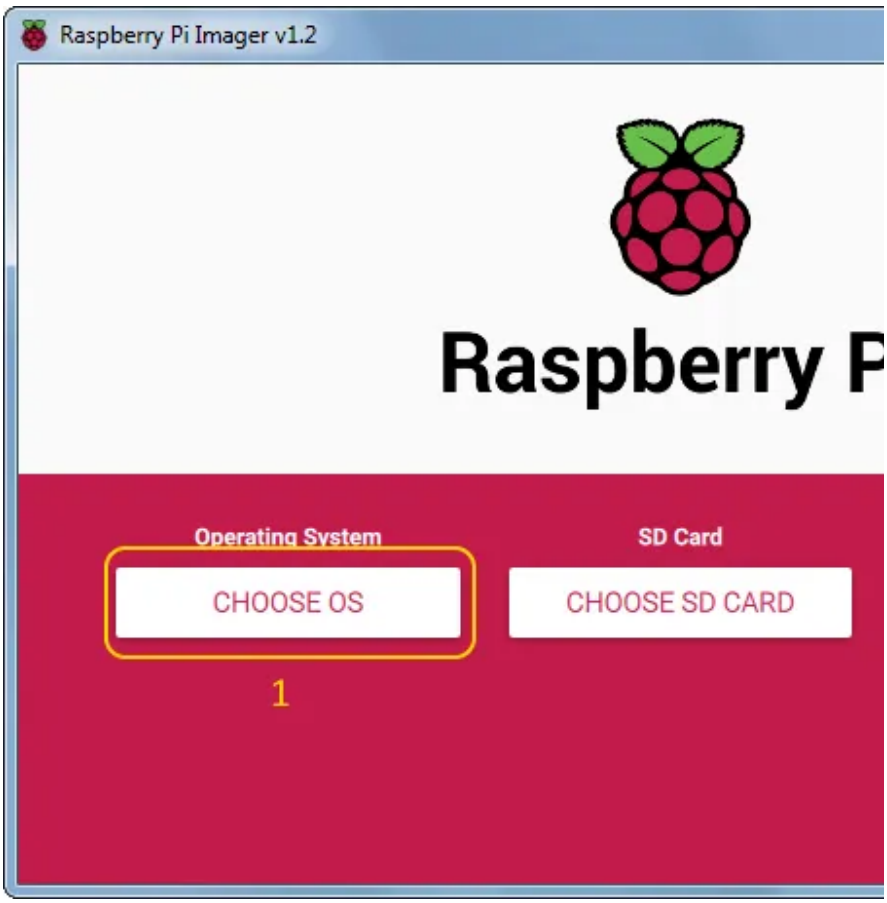
Operating system.

Use a fast SD memory card for your Raspberry Pi 4. 100MB/s and a capacity of 32 GB is perfect.

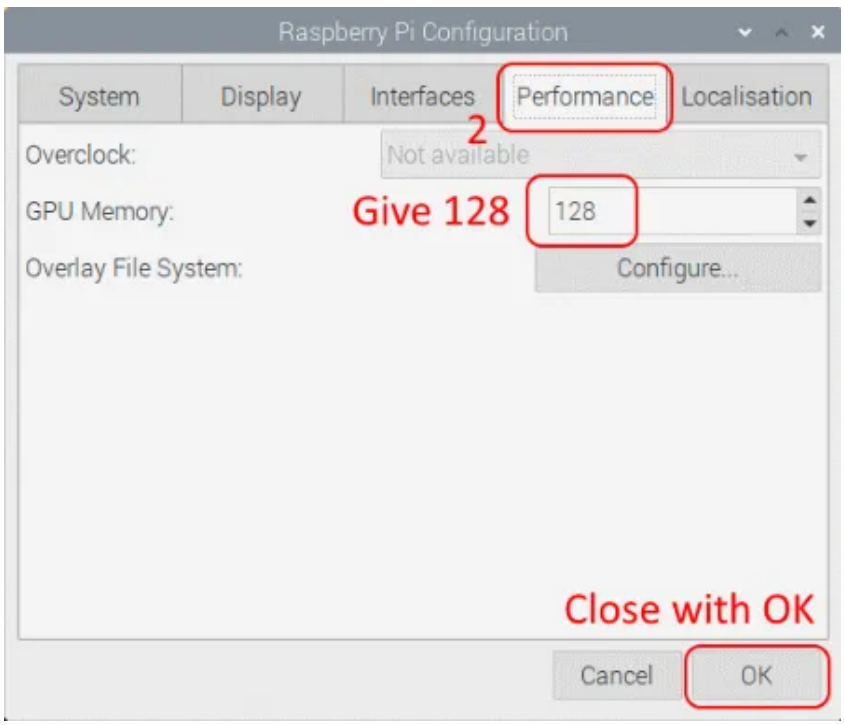
The operating system is installed by the new Raspberry Pi Image Tool from this site. The Image Tool can write an operating system of your choice on an SD card. At the same time, it will format the card into the correct ext4 for the Raspberry Pi, even a 64, 128 or even 256 GByte card. Different image formats are supported, making this tool a better alternative to balena Etcher.

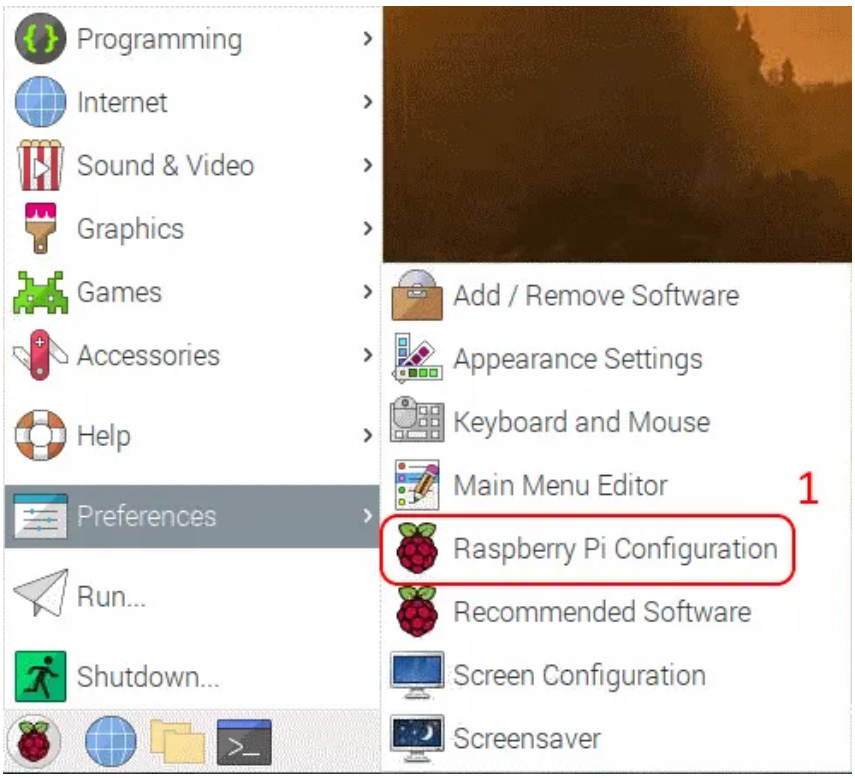
The second step is downloading the OS of your choice for your Raspberry Pi 4. The slideshow below shows how easy the whole image writing process is. After successful installing the Raspbian operating system, it is time to update and upgrade your operating system with the next commands in the terminal.





As explained here, the physical RAM chip is used both by the CPU and the GPU. On a Raspberry Pi 2 or 3 default is 64 Mbyte allocated for the GPU. The Raspberry Pi 4 has a 76 Mbyte GPU memory size. It can be somewhat small for vision projects, better to change this now to a 128 Mbyte. To increase the amount of memory for the GPU, use the following menu. After this action, the system wants to reboot. Let's do that.





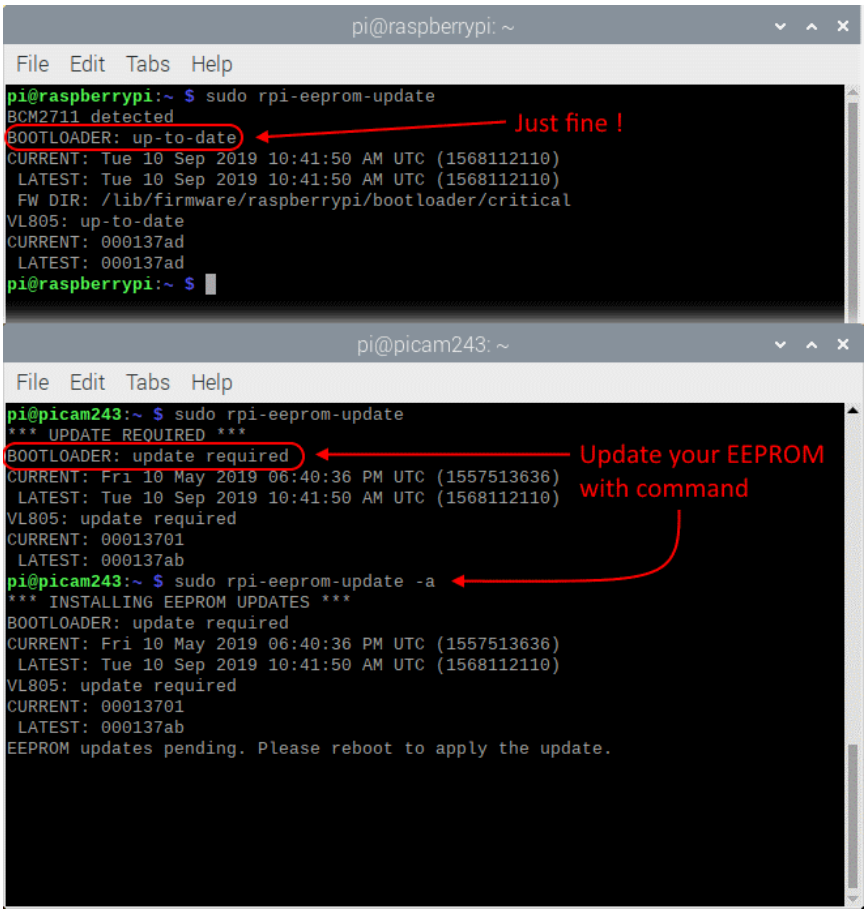
EEPROM.

With a fresh and clean Raspbian operating system, 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 Pi 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
```

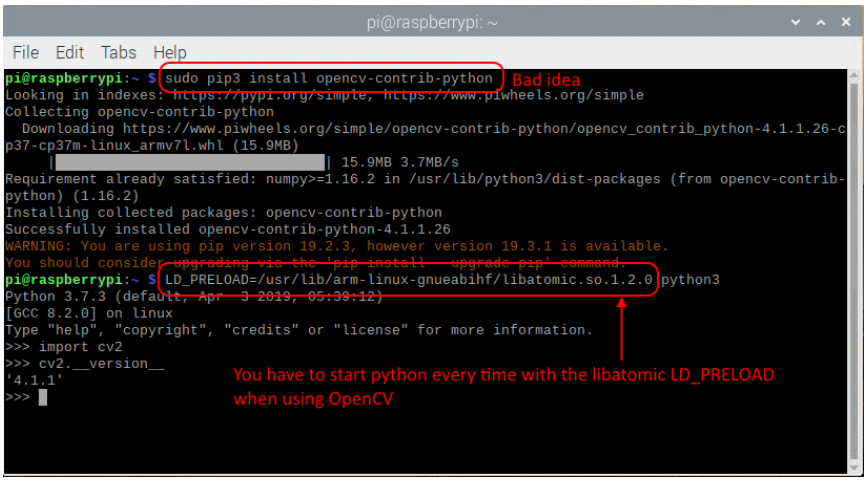
```
# if needed, to update the firmware
$ sudo rpi-eeprom-update -a

$ reboot
```



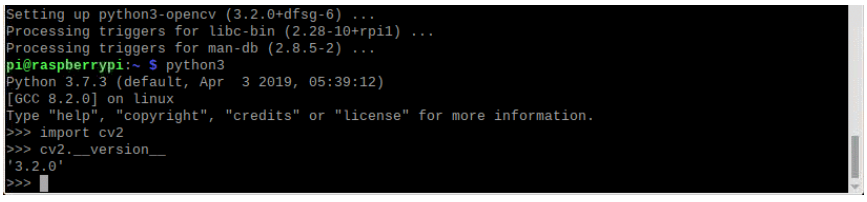
Bad ideas.

Some words of warning. Do not use pip to install OpenCV on your Raspberry Pi. First of all, pip installations don't support C++ due to missing header files. If you want to write code in C++, as we like to do, never use pip. Secondly, at the time of writing (April 2020), the OpenCV 4.1.1 version will be installed by pip. This version depends on an atomic library. Although available, the Raspbian operation system will only use this library when forced to do so. This can be done by the LD_PRELOAD trick. Not a convenient way to have this explicit

dependency. And as of last, with NOOBS 3.1.1, you will be dragged into missing dependencies like libQTtest etc.



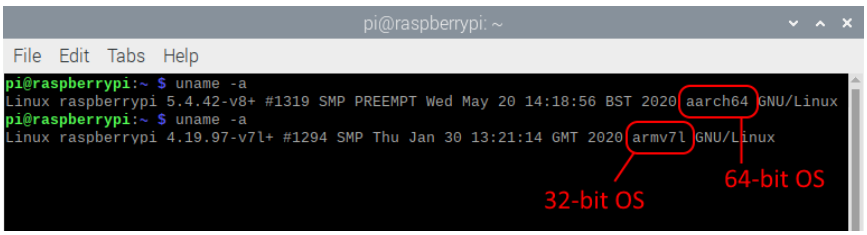
The same story applies to sudo apt-get install python3-opencv. The installed version of OpenCV (3.2.0 at the moment) neither supports C++. Another possible pitfall may be the repository. As soon as this ecosystem is updated with a newer version, a simple command like sudo apt-get upgrade will automatically install version 4.0.0 or higher which is incompatible with the 3 series. Unless you have isolated your OpenCV in a virtual environment, you are facing some time-consuming repair work on your old projects. On top of this all, and even worse, the installed version is not optimized for the NEON-ARM cores of the Raspberry Pi. So you end up with a library that could be much faster if it was installed manually according to the procedure below.



The sudo apt-get install libopencv-dev installation is another not so good idea. It generates a set of files suitable for C ++ coding. However, this OpenCV package is not recognized by Python. The installed version is again the old-fashioned 3.2.0. At the risk that once the ecosystem is updated, a sudo apt-get upgrade command will replace it for the incompatible 4.0.0 version.

Version check.

Before you install OpenCV on your Raspberry Pi 4, it is time for a final version check. Many readers just jump into the guide, skipping the introduction, often because they have already an operating system working. For those, please give the command `uname -a` and check your version.



Do you have the 32-bit version, `armv7l`, please continue. If your version is the 64 bit, `aarch64`, you can follow the instructions on this page.

Curious about the Raspberry 64 bit operating system? Check this page to see if an upgrade may be an idea.

Dependencies.

The OpenCV software uses other third-party software libraries. These have to be installed first. Some come with the Raspbian operating system, others may have been gathered over time, but it's better to be safe than sorry, so here is the complete list. Only the latest packages are installed by the procedure.

```
$ sudo apt-get update

$ sudo apt-get upgrade

$ sudo apt-get install cmake gfortran

$ sudo apt-get install libjpeg-dev libtiff-
dev libgif-dev


$ sudo apt-get install libavcodec-dev
libavformat-dev libswscale-dev

$ sudo apt-get install libgtk2.0-dev
libcanberra-gtk*

$ sudo apt-get install libxvidcore-dev
libx264-dev libgtk-3-dev

$ sudo apt-get install libtbb2 libtbb-dev
libdc1394-22-dev libv4l-dev

$ sudo apt-get install libopenblas-dev
libatlas-base-dev libblas-dev

$ sudo apt-get install libjasper-dev
liblapack-dev libhdf5-dev

$ sudo apt-get install gcc-arm* 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.

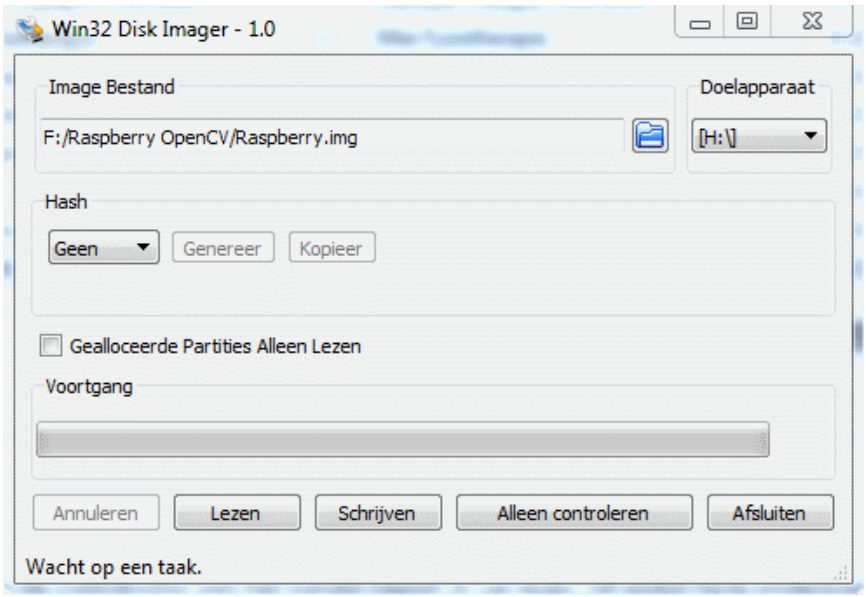
Now it is time to decide whether or not to use a virtual environment for your OpenCV installation. One of the main reasons for applying virtual environments is to protect your projects against incompatibility between package versions. Suppose you have OpenCV 2.4.13 and OpenCV 4.4.0 installed on your Raspberry Pi and you give the command `>> import cv2` in Python 3. Which version will Python take? Placing the versions in separate environments is the solution to this problem. Now you can activate one environment with the command `workon cv2413`, do something with the OpenCV 2.4.13 library, leave environment with `deactivate`, and enter the OpenCV 4.4.0 environment with workon `cv430` for instance.

If you stick to one OpenCV version, as many people do, there is no need for a virtual environment. Neither if you program in C++. In this language you must explicitly tell the compiler where the OpenCV folders with the headers and libraries are located. Obvious, you have installed your versions in unique locations. By the way, a bare OpenCV takes up about 1 Gbyte of space on your SD card.

Another way of dealing with versions is by using different micro-SD cards, each with a complete Raspbian OS and a different OpenCV version. Now you don't have different virtual

environments, but separate hardware environments. The price of those micro-SD cards is nowadays certainly no obstacle.

Speaking of SD cards. You can back up an image of your SD card before you upgrade to a newer OpenCV version. If you facing incompatibility issues, you can now quickly return to your previous work. By the way, it's always a good idea to make a complete backup of your SD card now and then. With Win32 Disk Imager it is done in no time.



Now it is up to you. If you want to install the virtual

environment software, follow the next steps. If not, skip this section and go further here.

Install a virtual environment.

Step one is some administration. We only use Python 3 because the support of Python 2.7 has stopped at the beginning of 2020. You have first to determine your Python 3 version and location. This location must be placed in the (hidden) ~/.bashrc file. The easiest way is a direct injection at the end of the file. You don't have to use an editor like nano or leafpad now.

```
# get version
$ python3 --version
# get location
$ which python 3.7
# merge VIRTUALENVWRAPPER_PYTHON=location/version
$ echo "export
VIRTUALENVWRAPPER_PYTHON=/usr/bin/python3.7"
>> ~/.bashrc

# reload profile
$ source ~/.bashrc
```

The Python location in the above commands was `/usr/bin/python3.7`. This location is passed as an argument in the `echo` command. The next step is installing the virtual environment software. This can be done with the following commands.

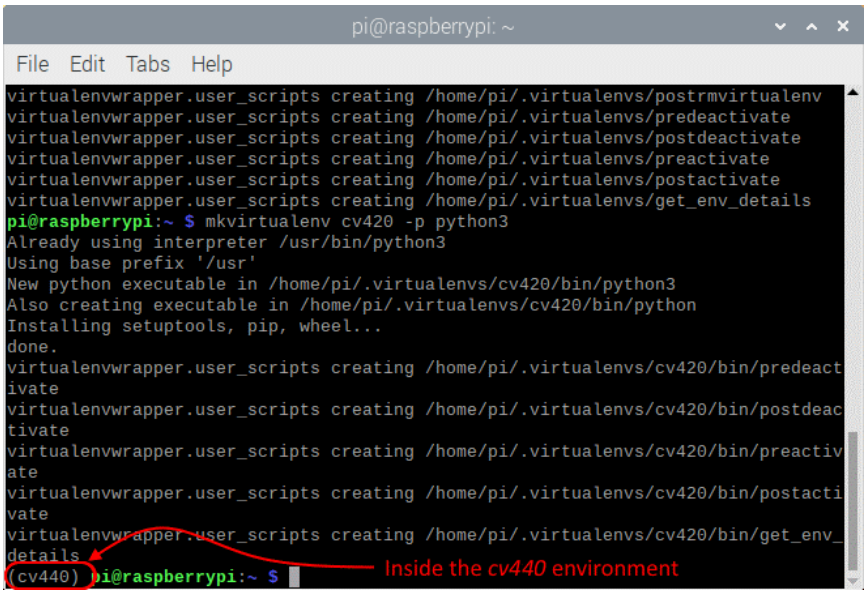
```
$ sudo pip3 install virtualenv
$ sudo pip3 install virtualenvwrapper
```

The last step is again some administration in the ~/.bashrc file, followed by a re-activation.

With the command `mkvirtualenv`, a virtual environment is set up for the OpenCV installation.

```
$ echo "export
WORKON_HOME=$HOME/.virtualenvs" >> ~/.bashrc

$ echo "source
/usr/local/bin/virtualenvwrapper.sh" >>
~/.bashrc

$ source ~/.bashrc

$ mkvirtualenv cv440
```

Your screen will be something like this.



There is only one dependency for Python with OpenCV in a virtual environment and that is Numpy. Although you have previously installed this package, you must also install it within the environment, otherwise, CMake cannot compile.

```
# without sudo!!!!
```

```
$ pip3 install numpy
```

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.

Recently there are some issues with Googles protobuf missing the link to the atomic library on the Raspberry Pi. When not found, it generates `undefined symbol: __atomic_fetch_add_8` errors. An extra linker flag was introduced in the CMake build: `OPENCV_EXTRA_EXE_LINKER_FLAGS=-latomic` to overcome the problem. According to ticket #5219 the issue is now solved. That's why dropped the extra linker flag from our build. If you still facing problems with the atomic library, incorporate the line and rebuild OpenCV after removing the whole build folder first.

```
$ cmake -D CMAKE_BUILD_TYPE=RELEASE \
        -D CMAKE_INSTALL_PREFIX=/usr/local \
        -D
OPENCV_EXTRA_MODULES_PATH=~/opencv_contrib/modules
\
        -D ENABLE_NEON=ON \
        -D ENABLE_VFPV3=ON \
        -D WITH_OPENMP=ON \
        -D BUILD_TIFF=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 ..
```

If everything went well, CMake generates a report that looks something like this (for readability purposes we omitted most lines). Very crucial are the Python sections. If these are missing, OpenCV will not install proper Python libraries. In that

case, usually CMake could not find the Python folders, or in the case of a virtual environment, a dependency such as numpy is probably not installed within the environment. We are experiencing these issues when removing too many packages from the operating system, as mentioned earlier. NEON and VFPV3 support must also be enabled. Certainly if you intend to build our deep learning examples. Check if `v4l/v4l2` is available if your planning to use the Raspicam.

```
-- General configuration for OpenCV 4.4.0
=====================================

--   Version control:               unknown

--

--   Extra modules:

--     Location (extra):            /home/pi/opencv_contrib/modules

--     Version control (extra):     unknown

--

--   Platform:

--     Timestamp:                   2020-07-29T12:40:57Z

--     Host:                        Linux 4.19.75-v7l+ armv7l

--     CMake:                       3.13.4

--     CMake generator:             Unix Makefiles

--     CMake build tool:            /usr/bin/make

--     Configuration:               RELEASE

--

--   CPU/HW features:

--     Baseline:                    VFPV3 NEON

--       requested:                 DETECT

--       required:                  VFPV3 NEON

--

--   C/C++:

--     Built as dynamic libs?:      YES

--     C++ Compiler:                /usr/bin/c++  (ver 8.3.0)

**********************

--     C Compiler:                  /usr/bin/cc

**********************

--     Documentation:               NO
```

```
--     Non-free algorithms:         YES

***********************

--   Video I/O:

--   DC1394:                      YES (2.2.5)

--   FFMPEG:                      YES

--     avcodec:                   YES (58.35.100)

--     avformat:                  YES (58.20.100)

--     avutil:                    YES (56.22.100)

--     swscale:                   YES (5.3.100)

--     avresample:                NO

--   GStreamer:                   NO

--   v4l/v4l2:                    YES (linux/videodev2.h)

--


--   Parallel framework:            TBB (ver 2020.2 interface
11102)

***********************


--   Python 2:

--     Interpreter:                 /usr/bin/python2.7 (ver 2.7.16)

--     Libraries:                   /usr/lib/arm-linux-
gnueabihf/libpython2.7.so (ver 2.7.16)

--     numpy:                       /usr/lib/python2.7/dist-
packages/numpy/core/include (ver 1.16.2)

--     install path:                lib/python2.7/dist-
packages/cv2/python-2.7

--

--   Python 3:

--     Interpreter:                 /usr/bin/python3 (ver 3.7.3)

--     Libraries:                   /usr/lib/arm-linux-
gnueabihf/libpython3.7m.so (ver 3.7.3)

--     numpy:                       /usr/lib/python3/dist-
packages/numpy/core/include (ver 1.16.2)

--     install path:                lib/python3.7/dist-
packages/cv2/python-3.7

--

--   Python (for build):            /usr/bin/python2.7

--

--   Java:

--     ant:                         /usr/bin/ant (ver 1.10.5)

--     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
```

Another issue we came across was the missing of the c++ compiler. CMake generated the screen dump below. By simply giving the Cmake command a second time, the problem was solved. In rare occasions, we had to `sudo apt-get update` and `upgrade` the system before CMake could find the c++ compiler.

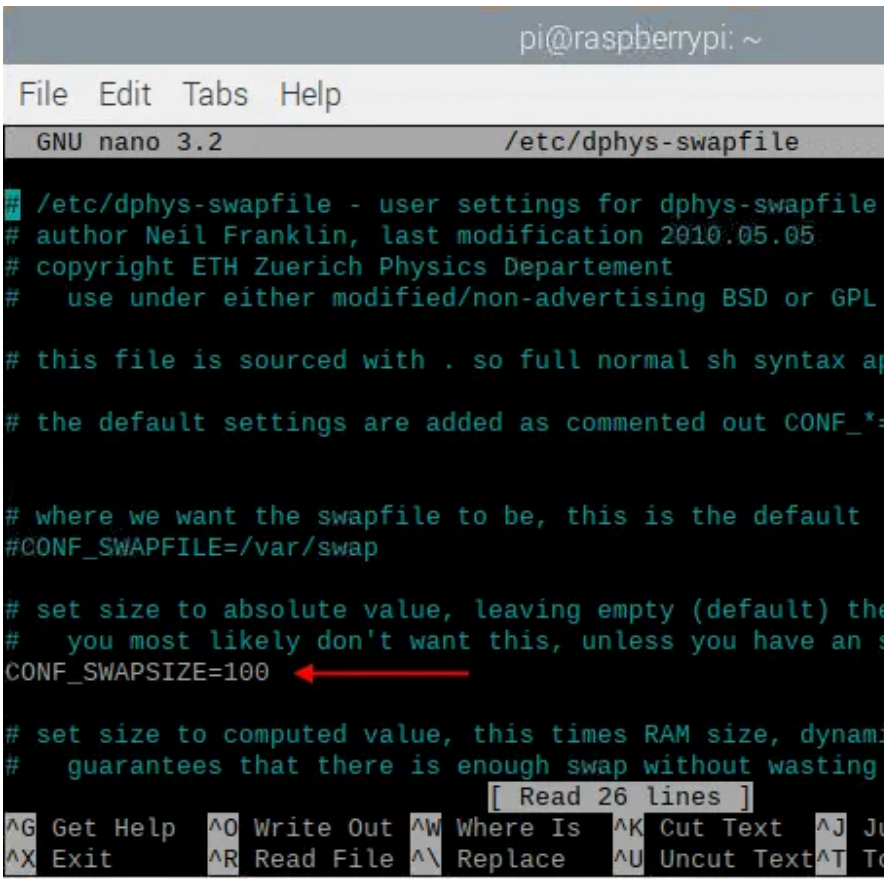
```
-- The CXX compiler identification is GNU 8.3.0
-- The C compiler identification is GNU 8.3.0
-- Check for working CXX compiler: /usr/bin/c++
-- Check for working CXX compiler: /usr/bin/c++ -- broken
CMake Error at /usr/share/cmake-
3.13/Modules/CMakeTestCXXCompiler.cmake:45 (message):
 The C++ compiler


   "/usr/bin/c++"


 is not able to compile a simple test program.
**********************
-- Configuring incomplete, errors occurred!
See also "/home/pi/opencv/build/CMakeFiles/CMakeOutput.log".
See also "/home/pi/opencv/build/CMakeFiles/CMakeError.log".
```

Before we can start the actual build, the memory swap space needs to be enlarged. For daily use a swap memory of 100

Mbyte is sufficient. However, with the massive build ahead of use, extra memory space is crucial. Enlarge the swap space with the following command.

```
$ sudo nano /etc/dphys-swapfile
```

This command opens Nano, a very lightweight text editor, with the system file `dphys-swapfile`. With the arrow keys, you can move the cursor to the CONF_SWAPSIZE line where the new value 2048 can be entered. Next, close the session with the <Ctrl+X> key combination. With <Y> and <Enter> changes are being saved in the same file.



Two additional commands are required before the new enlarge swap space is active.

```
$ sudo /etc/init.d/dphys-swapfile stop
$ sudo /etc/init.d/dphys-swapfile start
```

Make.

Now everything is ready for the great build. This takes a lot of time. Be very patient is the only advice here. Don't be surprised if at 99% your build seems to be crashed. That is 'normal' behaviour. Even when your CPU Usage Monitor gives very low

ratings like 7%. In reality, your CPU is working so hard it has not enough time to update these usage numbers correctly.

You can speed things up with four cores working simultaneously (`make -j4`). On a Raspberry Pi 4, it takes just over an hour to build the whole library. Sometimes the system crashes for no apparent reason at all at 99% or even 100%. In that case, restart all over again, as explained at the end of this page, and rebuild with `make -j1`.

Probably you get a whole bunch of warnings during the make. Don't pay to much attention to it. They are generated by subtle differences in template overload functions due to little version differences. So take coffee and a good book for reading, and start building with the next command.

```
$ make -j4
```

make -j4	Time (HH:MM)
Raspberry Pi 2 (900 MHz - 1 MB)	5:15
Raspberry Pi 3 (1400 MHz - 1 MB)	2:20

Raspberry Pi 4 (1500 MHz - 2 MB)	1:05

The times above are indications. No other processes were running like VNC or SSH. The memory swap was 2048 and there was no throttling (automatic lowering of the clock speed to prevent overheating). Hopefully, your build was as successful as the one below.

```
File  Edit  Tabs  Help
[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
```

There is one thing left to do before the installation of OpenCV 4.4 on your Raspberry Pi 4 is completed. That is resetting the swap space back to its original 100 Mbyte. Flash memory can only write a limited number of cycles. In the end, it will wear your SD card out. It is therefore wise to keep memory swapping to a minimum. Besides, it also slows down your application. The final commands are deleting the now useless zip files and rebooting your system so that all changes are implemented..

```
$ sudo nano /etc/dphys-swapfile

set CONF_SWAPSIZE=100 with the Nano text editor

$ cd ~
$ rm opencv.zip
$ rm opencv_contrib.zip
$ reboot
```

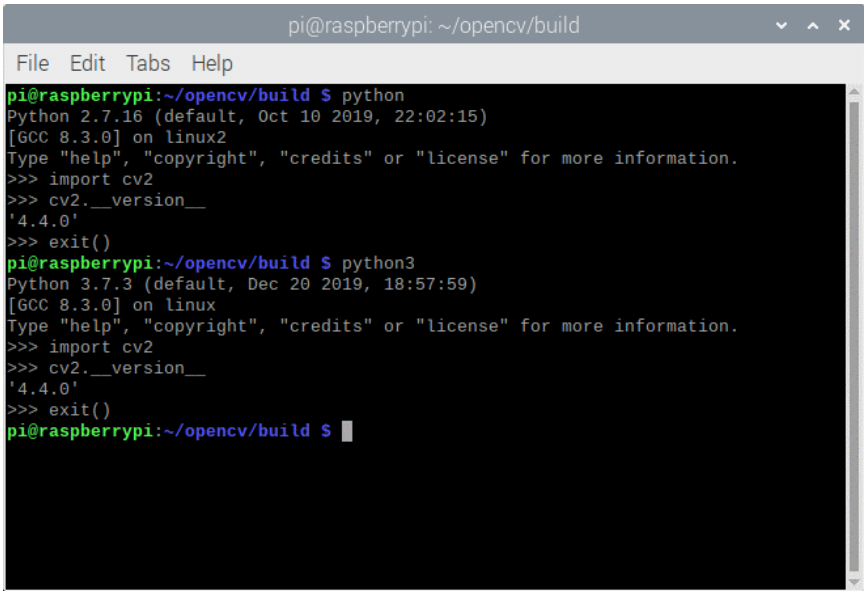
If you have installed OpenCV in a virtual environment, you need to make a symbolic link to the library. Without this link, OpenCV will not be found by python and the import fails. You can skip these steps if you have installed OpenCV without a virtual environment.

```
$ cd ~/.virtualenvs/cv420/lib/python3.7/site-packages
$ ln -s /usr/local/lib/python3.7/site-packages/cv2/python-3.7/cv2.cpython-37m-arm-linux-gnueabihf.so
$ cd ~
```

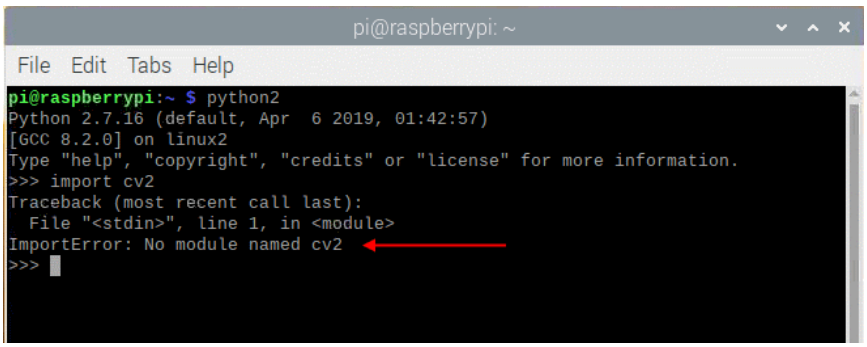
Checking

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`.



In rare occasions, Python can not find the OpenCV package. It comes with the message: “No module named cv2”.



There are many possible reasons, but most likely you started your build with the wrong version number of the library name in the CMake command (libpython3.7m.so). Another common cause is CMake not finding both Python versions. See your CMake rapport for this. You can also take a look at your OpenCV directory and copy the file manually to the correct

OpenCV directory and copy the file manually to the correct location.

```
$ cd ~/opencv/build/lib/python3
$ sudo cp cv2.cpython-37m-arm-linux-gnueabihf.so \
 /usr/local/lib/python3.7/dist-packages/cv2.so
```

It is always possible to alter some -D switch in CMake and rebuild the package. The already build modules are kept, only the new ones are generated. Of course, in the end, you may be facing still the long wait when everything must be linked into one large Python file. If you want a clean build, remove the complete build directory with all its subdirectories by the single command below. Next, make a new directory with the same name and start all over again.

```
$ cd ~/opencv
$ sudo rm -r build
```

Build information.

If you want to know in the future which software modules are included in your OpenCV build, you can always find the answer with the following Python command.

```
$ python
>>> import cv2
>>> print cv2.getBuildInformation()
```

Or use the following C++ code example.

```
#include <opencv2/opencv.hpp>

int main(void)
{
    std::cout << "OpenCV version : " << cv::CV_VER
    std::cout << "Major version : " << cv::CV_MAJO
    std::cout << "Minor version : " << cv::CV_MINO
    std::cout << "Subminor version : " << cv::CV_S
    std::cout << cv::getBuildInformation() << std:
}
```

And you get the next outcome.

```
General configuration for OpenCV 4.4.0
=====================================
 Version control:               unknown

 Extra modules:
   Location (extra):            /home/pi/opencv_contrib/modules
   Version control (extra):     unknown

 Platform:
   Timestamp:                   2020-07-29T12:40:57Z
   Host:                        Linux 5.4.51-v7l+ armv7l
   CMake:                       3.13.4
   CMake generator:             Unix Makefiles
   CMake build tool:            /usr/bin/make
   Configuration:               RELEASE

 CPU/HW features:
   Baseline:                    VFPV3 NEON
     requested:                 DETECT
     required:                  VFPV3 NEON

 C/C++:
   Built as dynamic libs?:      YES
   C++ standard:                11
   C++ Compiler:                /usr/bin/c++  (ver 8.3.0)
   C++ flags (Release):         -fsigned-char -W -Wall -
```

```
   C++ flags (Release):         -fsigned-char -W -Wall -
Werror=return-type -Werror=non-virtual-dtor -Werror=address -
Werror=sequence-point -Wformat -Werror=format-security -Wmissing-
declarations -Wundef -Winit-self -Wpointer-arith -Wshadow -Wsign-
promo -Wuninitialized -Winit-self -Wsuggest-override -Wno-delete-
non-virtual-dtor -Wno-comment -Wimplicit-fallthrough=3 -Wno-strict-
overflow -fdiagnostics-show-option -pthread -fomit-frame-pointer -
ffunction-sections -fdata-sections  -mfpu=neon -fvisibility=hidden
-fvisibility-inlines-hidden -fopenmp -O3 -DNDEBUG  -DNDEBUG

   C++ flags (Debug):           -fsigned-char -W -Wall -
Werror=return-type -Werror=non-virtual-dtor -Werror=address -
Werror=sequence-point -Wformat -Werror=format-security -Wmissing-
declarations -Wundef -Winit-self -Wpointer-arith -Wshadow -Wsign-
promo -Wuninitialized -Winit-self -Wsuggest-override -Wno-delete-
non-virtual-dtor -Wno-comment -Wimplicit-fallthrough=3 -Wno-strict-
overflow -fdiagnostics-show-option -pthread -fomit-frame-pointer -
ffunction-sections -fdata-sections  -mfpu=neon -fvisibility=hidden
-fvisibility-inlines-hidden -fopenmp -g  -O0 -DDEBUG -D_DEBUG

   C Compiler:                  /usr/bin/cc

   C flags (Release):           -fsigned-char -W -Wall -
Werror=return-type -Werror=non-virtual-dtor -Werror=address -
Werror=sequence-point -Wformat -Werror=format-security -Wmissing-
declarations -Wmissing-prototypes -Wstrict-prototypes -Wundef -
Winit-self -Wpointer-arith -Wshadow -Wuninitialized -Winit-self -
Wno-comment -Wimplicit-fallthrough=3 -Wno-strict-overflow -
fdiagnostics-show-option -pthread -fomit-frame-pointer -ffunction-
sections -fdata-sections  -mfpu=neon -fvisibility=hidden -fopenmp -
O3 -DNDEBUG  -DNDEBUG

   C flags (Debug):             -fsigned-char -W -Wall -
Werror=return-type -Werror=non-virtual-dtor -Werror=address -
Werror=sequence-point -Wformat -Werror=format-security -Wmissing-
declarations -Wmissing-prototypes -Wstrict-prototypes -Wundef -
Winit-self -Wpointer-arith -Wshadow -Wuninitialized -Winit-self -
Wno-comment -Wimplicit-fallthrough=3 -Wno-strict-overflow -
fdiagnostics-show-option -pthread -fomit-frame-pointer -ffunction-
sections -fdata-sections  -mfpu=neon -fvisibility=hidden -fopenmp -
g  -O0 -DDEBUG -D_DEBUG

   Linker flags (Release):      -Wl,--gc-sections -Wl,--as-needed

   Linker flags (Debug):        -Wl,--gc-sections -Wl,--as-needed

   ccache:                      NO

   Precompiled headers:         NO

   Extra dependencies:          dl m pthread rt

   3rdparty dependencies:


 OpenCV modules:

   To be built:                 aruco bgsegm bioinspired calib3d
ccalib core datasets dnn dnn_objdetect dnn_superres dpm face
features2d flann freetype fuzzy gapi hdf hfs highgui img_hash
imgcodecs imgproc intensity_transform line_descriptor ml objdetect
optflow phase_unwrapping photo plot python2 python3 quality rapid
reg rgbd saliency shape stereo stitching structured_light superres
surface_matching text tracking ts video videoio videostab
xfeatures2d ximgproc xobjdetect xphoto

   Disabled:                    world
```

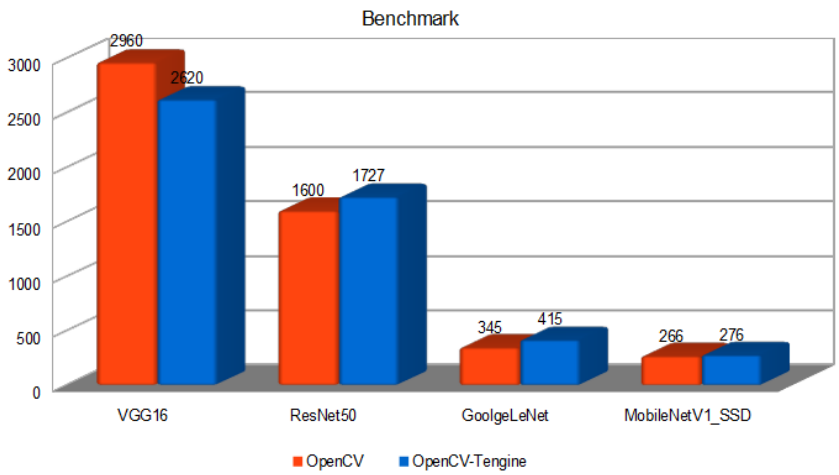
```
   Disabled:                    world

   Disabled by dependency:      -

   Unavailable:                 alphamat cnn_3dobj cudaarithm
cudabgsegm cudacodec cudafeatures2d cudafilters cudaimgproc
cudalegacy cudaobjdetect cudaoptflow cudastereo cudawarping cudev
cvv java js julia matlab ovis sfm viz

   Applications:                perf_tests apps

   Documentation:               NO

   Non-free algorithms:         YES


 GUI:

   GTK+:                        YES (ver 3.24.5)

     GThread :                  YES (ver 2.58.3)

     GtkGlExt:                  NO


 Media I/O:

   ZLib:                        /usr/lib/arm-linux-
gnueabihf/libz.so (ver 1.2.11)

   JPEG:                        /usr/lib/arm-linux-
gnueabihf/libjpeg.so (ver 62)

   WEBP:                        build (ver encoder: 0x020f)

   PNG:                         /usr/lib/arm-linux-
gnueabihf/libpng.so (ver 1.6.36)

   TIFF:                        build (ver 42 - 4.0.10)

   JPEG 2000:                   /usr/lib/arm-linux-
gnueabihf/libjasper.so (ver 1.900.1)

   OpenEXR:                     build (ver 2.3.0)

   HDR:                         YES

   SUNRASTER:                   YES

   PXM:                         YES

   PFM:                         YES


 Video I/O:

   DC1394:                      YES (2.2.5)

   FFMPEG:                      YES

     avcodec:                   YES (58.35.100)

     avformat:                  YES (58.20.100)

     avutil:                    YES (56.22.100)

     swscale:                   YES (5.3.100)

     avresample:                NO

   GStreamer:                   NO

   v4l/v4l2:                    YES (linux/videodev2.h)
```

```
  Parallel framework:            TBB (ver 2020.2 interface 11102)

  Trace:                         YES (with Intel ITT)

  Other third-party libraries:
    Tengine:                     YES
(/home/pi/tengine/core/lib/libtengine.a)
    Lapack:                      NO
    Custom HAL:                  YES (carotene (ver 0.0.1))
    Protobuf:                    build (3.5.1)

  OpenCL:                        YES (no extra features)
    Include path:
                /home/pi/opencv/3rdparty/include/opencl/1.2
    Link libraries:              Dynamic load

  Python 2:
    Interpreter:                 /usr/bin/python2.7 (ver 2.7.16)
    Libraries:                   /usr/lib/arm-linux-
gnueabihf/libpython2.7.so (ver 2.7.16)
    numpy:                       /usr/lib/python2.7/dist-
packages/numpy/core/include (ver 1.16.2)
    install path:                lib/python2.7/dist-
packages/cv2/python-2.7

  Python 3:
    Interpreter:                 /usr/bin/python3 (ver 3.7.3)
    Libraries:                   /usr/lib/arm-linux-
gnueabihf/libpython3.7m.so (ver 3.7.3)
    numpy:                       /usr/lib/python3/dist-
packages/numpy/core/include (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
-----------------------------------------------------------------
```

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, 32-bit OS at 1500 MHz, timing in mSec.



← Camera OpenCV C++ examples

Deep learning examples for →

C++ examples examples for