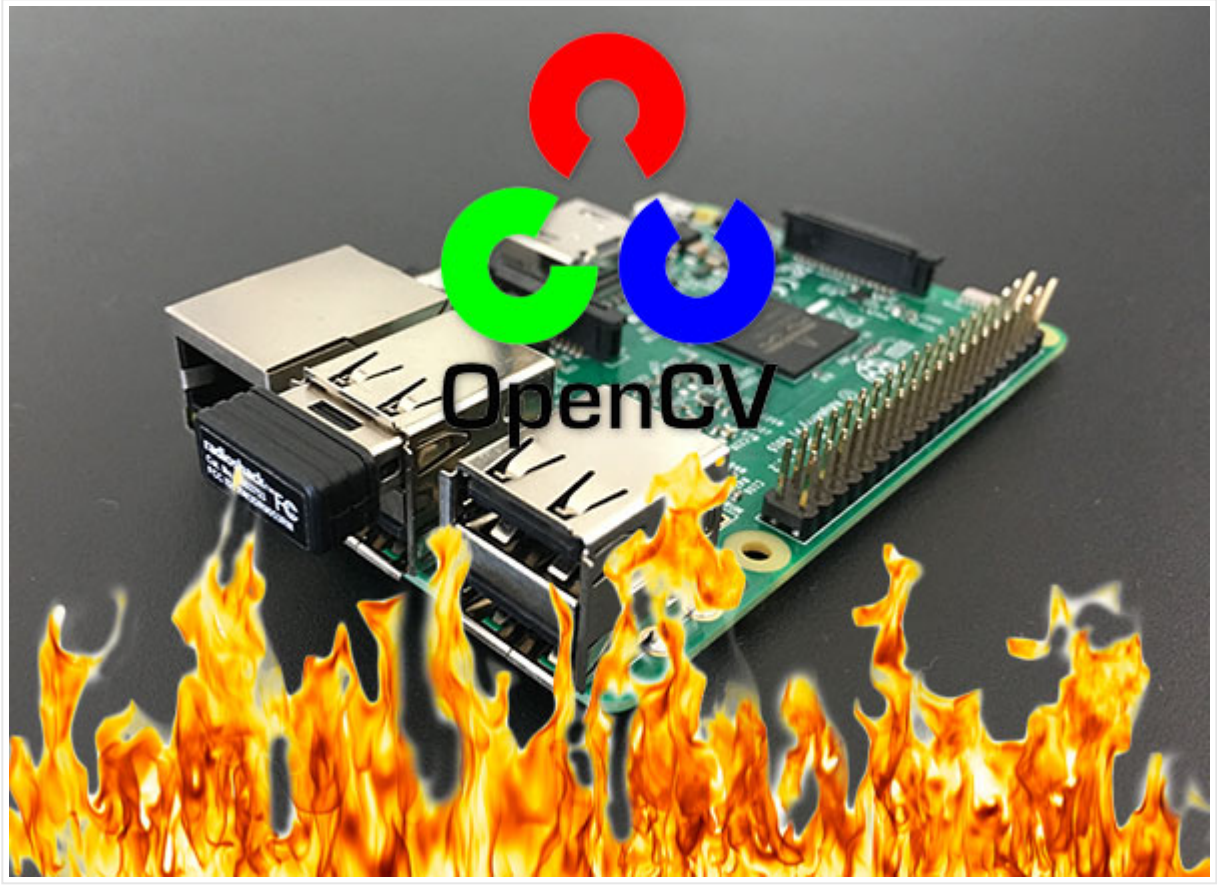
Optimizing OpenCV on the Raspberry Pi

by **Adrian Rosebrock** on October 9, 2017 in **Deep Learning**, **OpenCV 3**, **Raspberry Pi**



This tutorial is meant for ***advanced Raspberry Pi users*** who are looking to *milk every last bit of performance* out of their Pi for computer vision and image processing using OpenCV.

I'll be assuming:

1. You have worked through my previous Raspberry Pi + OpenCV install tutorials (ideally *multiple* times).
2. You are comfortable with the command line and Unix environments.

Since this is an advanced guide, I'll be doing less hand holding and instead focusing on the *optimizations themselves.* If you get stuck or run into an error, you'll need to refer back to the previous tutorials I offer here on PyImageSearch.

By the time you finish this tutorial, your Raspberry Pi will enjoy a ***30% speed increase*** when executing OpenCV and Python scripts.

To learn more about optimizing OpenCV on your Raspberry Pi, *just keep reading.*

Looking for the source code to this post?
Jump right to the downloads section.

Optimizing OpenCV on the Raspberry Pi

A couple weeks ago I demonstrated how to deploy a deep neural network to your Raspberry Pi.

The results were satisfactory, taking approximately 1.7 seconds to classify an image using GoogLeNet and 0.9 seconds for SqueezeNet, respectively.

However, I was left wondering if we could do better.

While we cannot ***train*** neural networks on the Raspberry Pi, we can ***deploy*** pre-trained networks to our Pi — provided we can optimize the Raspberry Pi sufficiently (and the network can fit into the limited memory of the Pi hardware).

In the remainder of this tutorial, we will discuss the optimizations we will leverage during our OpenCV installation, followed by walking through the seven installation steps.

After our optimized OpenCV compile is installed, we'll run a few quick tests determine if our new OpenCV install is faster than the previous one.

My goal here to demonstrate that the optimizations are in fact ***much faster on the Raspberry Pi 3*** and you should not hesitate to use them in your own projects.

NEON and FVPV3

In my research on how to optimize the Raspberry Pi for OpenCV I came across this excellent article by Sagi Zeevi.

Inside the tutorial Sagi recommends using:

1. NEON
2. VFPV3
3. And optionally Threading Building Blocks (TBB)

I'm not a big fan of TBB as (1) the performance gains are meager and (2) they are a royal pain in the ass to install on the Raspberry Pi.

The most bang for your buck is going to come from NEON and VFPV3.

ARM NEON is an optimization architecture extension for ARM processors. It was designed by the ARM engineers specifically for faster video processing, image processing, speech recognition, and machine learning. This optimization supports Single Instruction Multiple Data (SIMD) (as opposed to SISD, MISD, MIMD), which describes an architecture where multiple processing elements in the pipeline perform operations on multiple data points (hardware) all executed with a single instruction.

The ARM engineers also built VFPV3, a floating point optimization, into the chip our Raspberry Pi 3's use. The ARM page linked here describes features included in this optimization such as configurable rounding modes and customizable default not a number (NaN) behavior.

What this means for us is that our neural network is likely to run faster because the ARM processor on the Raspberry Pi 3 has hardware optimizations that we can take advantage of with the 4× ARM Cortex-A53, 1.2GHz processor.

I think you'll be really impressed with the results, so let's go ahead and get your optimized OpenCV installed on the Raspberry Pi.

Step #1: Expand filesystem and reclaim space

For the remainder of this tutorial I'll be making the following assumptions:

1. You are working with a brand new, fresh install of **Raspbian Stretch**.
2. This is not the first time you have installed OpenCV on the Raspberry Pi using Python virtual environments. If it is, please get your feet wet using one of my introductory OpenCV install guides.
3. You are comfortable with the command line and Unix environments.
4. You know how to debug CMake output for common errors (Python virtual environment not found, missing Python libraries, etc.).

Again, this tutorial is an **advanced guide** so I'll presenting the commands and only providing an explanation if it is pertinent — by and large, you should know what these commands do before you execute them.

The first step is to run, `raspi-config` and expand your filesystem:

```
Optimizing OpenCV on the Raspberry Pi                    Shell
1 $ sudo raspi-config
```

And then reboot your Raspberry Pi:

```
Optimizing OpenCV on the Raspberry Pi                    Shell
1 $ sudo reboot
```

From there, delete both Wolfram Engine and LibreOffice to reclaim ~1GB of space on your Raspberry Pi:

```
Optimizing OpenCV on the Raspberry Pi                    Shell
1 $ sudo apt-get purge wolfram-engine
2 $ sudo apt-get purge libreoffice*
3 $ sudo apt-get clean
4 $ sudo apt-get autoremove
```

Step #2: Install dependencies

The following commands will update and upgrade any existing packages, followed by installing dependencies, I/O libraries, and optimization packages for OpenCV:

```
Optimizing OpenCV on the Raspberry Pi                    Shell
1 $ sudo apt-get update && sudo apt-get upgrade
```

```
$ sudo apt-get install build-essential cmake 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 libgtk2.0-dev libgtk-3-dev
$ sudo apt-get install libcanberra-gtk*
$ sudo apt-get install libatlas-base-dev gfortran
$ sudo apt-get install python2.7-dev python3-dev
```

This entire process should take about 5 minutes.

***Note:** I added `libcanberra-gtk*` which grabs the ARM specific GTK to prevent GTK warnings (not errors; warnings) you may encounter when running Python + OpenCV scripts on the Raspberry Pi.*

Step #3: Download the OpenCV source code

Next, download the OpenCV source code for both the opencv and opencv_contrib repositories, followed by unarchiving them:

```
Optimizing OpenCV on the Raspberry Pi                    Shell
$ cd ~
$ wget -O opencv.zip https://github.com/Itseez/opencv/archive/3.3
$ unzip opencv.zip
$ wget -O opencv_contrib.zip https://github.com/Itseez/opencv_con
$ unzip opencv_contrib.zip
```

***Note:** You will need to click the "<=>" button in the toolbar of the codeblock above to grab the full paths to the zip archives.*

For this blog post, we'll be using **OpenCV 3.3**; however, as newer versions of OpenCV are released you can update the corresponding version numbers.

Step #4: Create your Python virtual environment and install NumPy

We'll be using Python virtual environments, a best practice when working with Python.

You can install pip, virtualenv, and virtualenvwrapper using the following commands:

```
Optimizing OpenCV on the Raspberry Pi                    Shell
$ wget https://bootstrap.pypa.io/get-pip.py
$ sudo python get-pip.py
$ sudo python3 get-pip.py
$ sudo pip install virtualenv virtualenvwrapper
$ sudo rm -rf ~/.cache/pip
```

Once both `virtualenv` and `virtualenvwrapper` have been installed, open up your `~/.profile` and append the following lines to the *bottom* of the file, using your favorite terminal-based text editor such as `vim`, `emacs`, or `nano`:

```
Optimizing OpenCV on the Raspberry Pi                    Shell
# virtualenv and virtualenvwrapper
export WORKON_HOME=$HOME/.virtualenvs
source /usr/local/bin/virtualenvwrapper.sh
```

From there, reload your `~/.profile` file to apply the changes to your current bash session:

```
Optimizing OpenCV on the Raspberry Pi                    Shell
$ source ~/.profile
```

You'll need to run `source ~/.profile` ***each time*** you open a new terminal/SSH into your Pi to ensure your system variables have been set properly (it also loads this file on boot).

Next, create your Python 3 virtual environment:

```
Optimizing OpenCV on the Raspberry Pi                    Shell
$ mkvirtualenv cv -p python3
```

Here I am creating a Python virtual environment named `cv` using Python 3 (alternatively, you may also use Python 2.7 by changing the `-p` switch to `python2`).

You can name the virtual environment whatever you want, but I use `cv` as the standard naming convention here on PyImageSearch.

Finally, install NumPy into the Python virtual environment:

```
Optimizing OpenCV on the Raspberry Pi                    Shell
$ pip install numpy
```

Step #5: Compile and install the *optimized* OpenCV library for Raspberry Pi

We're now ready to compile and install the optimized version of Raspberry Pi.

Ensure you are in the `cv` virtual environment using the `workon` command:

```
Optimizing OpenCV on the Raspberry Pi                    Shell
$ workon cv
```

And from there configure your build:

```
Optimizing OpenCV on the Raspberry Pi                    Shell
$ cd ~/opencv-3.3.0/
$ mkdir build
$ cd build
$ cmake -D CMAKE_BUILD_TYPE=RELEASE \
    -D CMAKE_INSTALL_PREFIX=/usr/local \
    -D OPENCV_EXTRA_MODULES_PATH=~/opencv_contrib-3.3.0/modules
    -D ENABLE_NEON=ON \
    -D ENABLE_VFPV3=ON \
    -D BUILD_TESTS=OFF \
    -D INSTALL_PYTHON_EXAMPLES=OFF \
    -D BUILD_EXAMPLES=OFF ..
```

Notice how the NEON and VFPV3 flags have been enabled.

If you're using Python 2.7, your *"Python 2"* section should look like this:

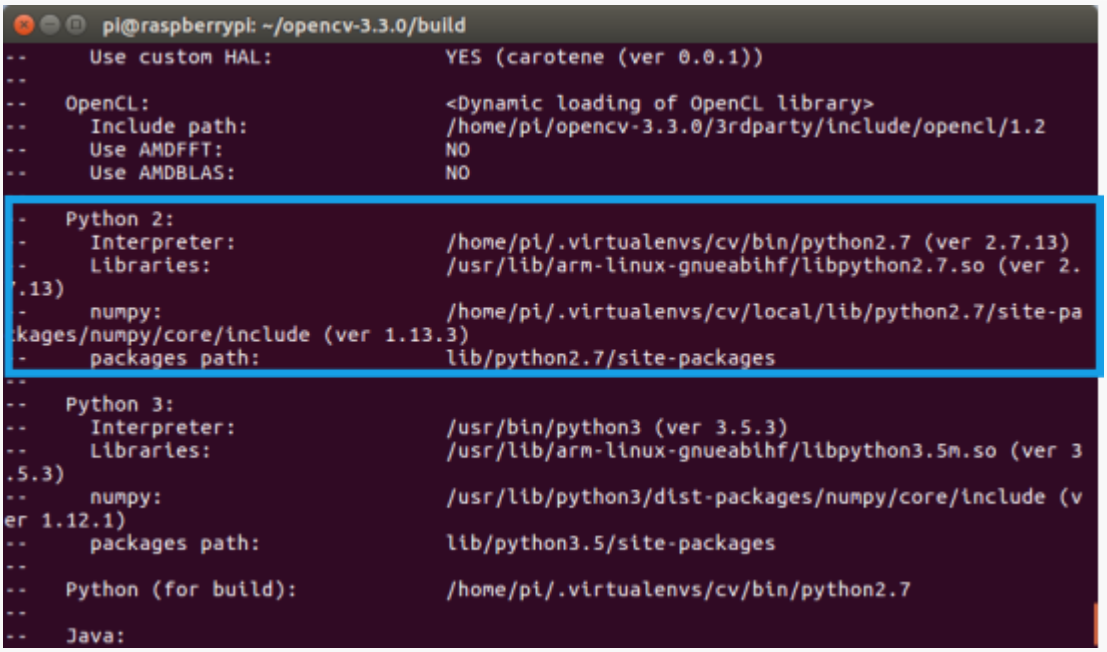


Figure 1: Running CMake to generate the build files for OpenCV 3.3. OpenCV will correctly be built with Python 2.7 and NumPy from our `cv` virtualenv.

Otherwise, if you're compiling OpenCV for Python 3, check the *"Python 3"*output of CMake:

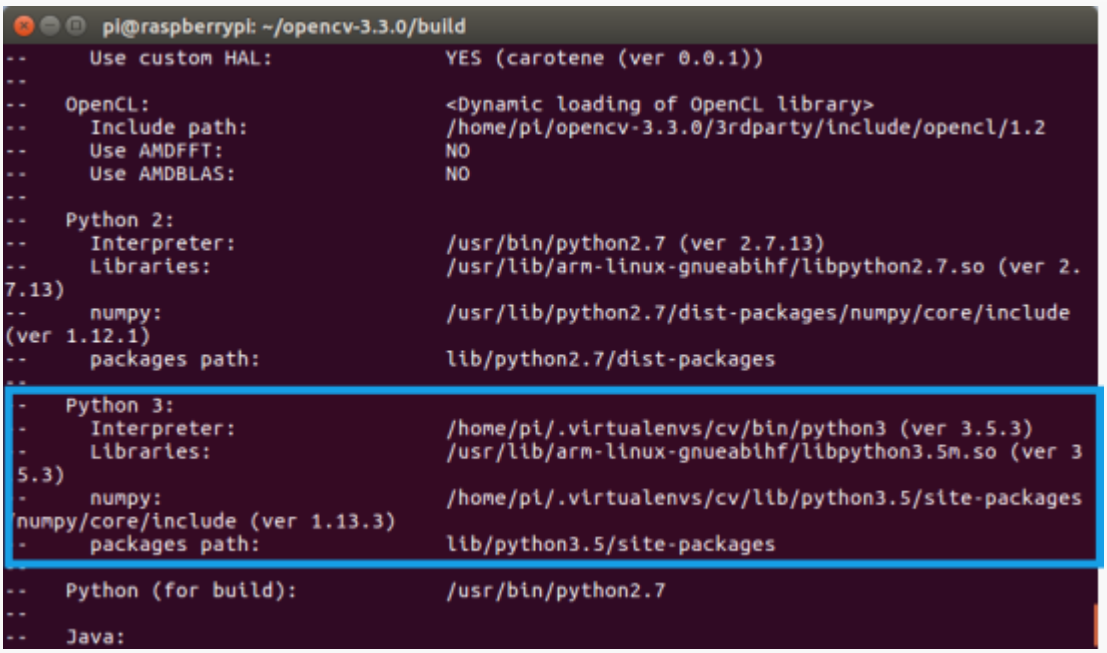


Figure 2: After running CMake, Python 3 + NumPy are correctly set from within our `cv`virtualenv on the Raspberry Pi.

Notice how the `Interpreter` , `Libraries` , `numpy` , and `packages path` variables have been properly set.

Before you start the compile I would suggest ***increasing your swap space***. This will enable you to compile OpenCV with ***all four cores*** of the Raspberry Pi without the compile hanging due to memory exhausting.

Open up your `/etc/dphys-swapfile` file and then edit the `CONF_SWAPSIZE` variable:

```
Optimizing OpenCV on the Raspberry Pi                    Shell
```

```
# set size to absolute value, leaving empty (default) then uses c
#   you most likely don't want this, unless you have an special d
# CONF_SWAPSIZE=100
CONF_SWAPSIZE=1024
```

Notice that I'm increasing the swap from 100MB to 1024MB. This is the secret sauce to compiling OpenCV with multiple cores on the Raspbian Stretch.

If you do not perform this step it's very likely that your Pi will hang.

From there, restart the swap service:

```
Optimizing OpenCV on the Raspberry Pi                    Shell
$ sudo /etc/init.d/dphys-swapfile stop
$ sudo /etc/init.d/dphys-swapfile start
```

Note: *Increasing swap size is a great way to burn out your Raspberry Pi microSD card. Flash-based storage have limited number of writes you can perform until the card is essentially unable to hold the 1's and 0's anymore. We'll only be enabling large swap for a short period of time, so it's not a big deal. Regardless, be sure to backup your* `.img` *file after installing OpenCV + Python just in case your card dies unexpectedly early. You can read more about large swap sizes corrupting memory cards on this page.*

Now that we've updated the swap size, kick off the optimized OpenCV compile using all four cores:

```
Optimizing OpenCV on the Raspberry Pi                    Shell
$ make -j4
```

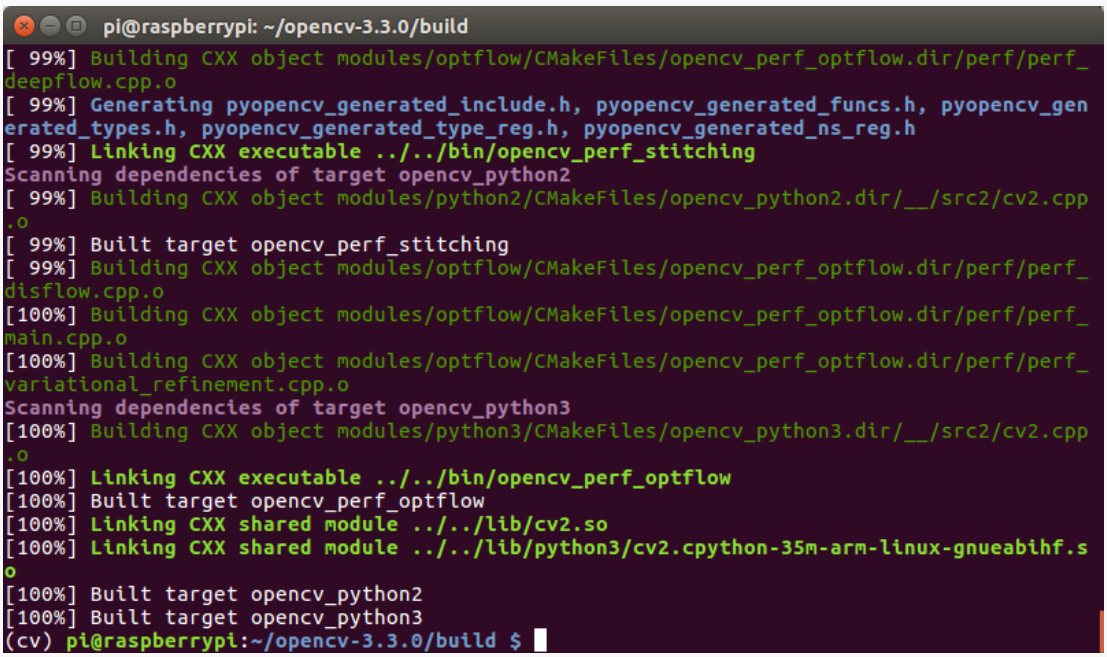


Figure 3: Our optimized compile of OpenCV 3.3 for the Raspberry Pi 3 has been completed successfully.

Assuming OpenCV compiled without error (as in my screenshot above), you can install your optimized version of OpenCV on your Raspberry Pi:

```
Optimizing OpenCV on the Raspberry Pi                    Shell
$ sudo make install
```

```
$ sudo ldconfig
```

Don't forget to go back to your `/etc/dphys-swapfile` file and:

1. Reset `CONF_SWAPSIZE` to 100MB.
2. Restart the swap service.

Step #6: Finish installing your *optimized* OpenCV on the Raspberry Pi

If you compiled ***OpenCV for Python 3***, you need to issue the following commands to sym-link the `cv2.so` bindings into your `cv` virtual environment:

```
Optimizing OpenCV on the Raspberry Pi                    Shell
$ cd /usr/local/lib/python3.5/site-packages/
$ sudo mv cv2.cpython-35m-arm-linux-gnueabihf.so cv2.so
$ cd ~/.virtualenvs/cv/lib/python3.5/site-packages/
$ ln -s /usr/local/lib/python3.5/site-packages/cv2.so cv2.so
```

Keep in mind that the exact paths will need to be updated depending if you are using Python 3.4, Python 3.5, Python 3.6, etc.

If you instead compiled **OpenCV for Python 2.7**, you can use these commands to sym-link your `cv2.so` file into the `cv` virtual environment:

```
Optimizing OpenCV on the Raspberry Pi                    Shell
$ cd ~/.virtualenvs/cv/lib/python2.7/site-packages/
$ ln -s /usr/local/lib/python2.7/site-packages/cv2.so cv2.so
```

Step 7: Testing your *optimized* OpenCV + Raspberry Pi install

As a quick sanity check, access the `cv` virtual environment, fire up a Python shell and try to import the OpenCV library:

```
Optimizing OpenCV on the Raspberry Pi                    Shell
$ source ~/.profile
$ workon cv
$ python
>>> import cv2
>>> cv2.__version__
'3.3.0'
>>>
```

Congratulations! You've just installed an optimized OpenCV 3.3 on your Raspberry Pi 3.

So, how good are these optimizations?

After working through this tutorial you're probably curious how good these OpenCV + Raspberry Pi optimizations are.

Given that we just optimized for floating point operations a great test would be to run a pre-trained deep neural network on the Raspberry Pi, similar to what we did last week.

Go ahead and use the ***"Downloads"*** section of this blog post to download our pre-trained Convolutional Neural Networks + example images + classification script.

From there, fire up a shell and execute the following command:

```
Optimizing OpenCV on the Raspberry Pi                                    Shell
$ python pi_deep_learning.py --prototxt models/bvlc_googlenet.pr
    --model models/bvlc_googlenet.caffemodel --labels synset_wor
    --image images/barbershop.png
[INFO] loading model...
[INFO] classification took 0.87173 seconds
[INFO] 1. label: barbershop, probability: 0.78055
[INFO] 2. label: barber chair, probability: 0.2194
[INFO] 3. label: rocking chair, probability: 3.4663e-05
[INFO] 4. label: restaurant, probability: 3.7258e-06
[INFO] 5. label: hair spray, probability: 1.4715e-06
```

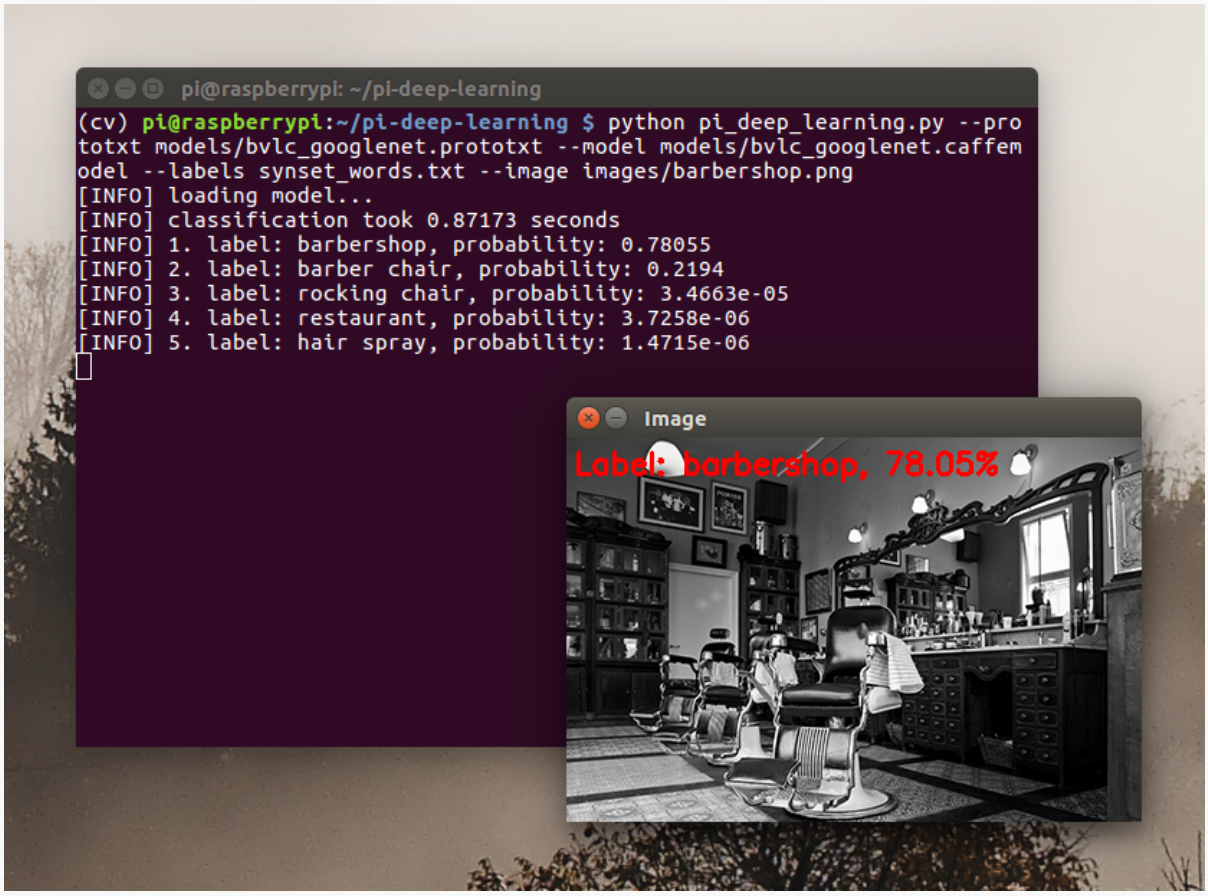


Figure 4: Running an image of a *"barbershop"* through GoogLeNet on the Raspberry Pi 3 with an optimized install of OpenCV 3.3 achieves a 48.82% speedup.

Here you can see that GoogLeNet classified our image in ***0.87 seconds***, which is a *massive* ***48.82%*** improvement from last week's 1.7 seconds.

Let's give SqueezeNet a try:

```
Optimizing OpenCV on the Raspberry Pi                                    Shell
$ python pi_deep_learning.py --prototxt models/squeezenet_v1.0.p
    --model models/squeezenet_v1.0.caffemodel --labels synset_wo
    --image images/barbershop.png
```

```
[INFO] loading model...
[INFO] classification took 0.4777 seconds
[INFO] 1. label: barbershop, probability: 0.80578
[INFO] 2. label: barber chair, probability: 0.15124
[INFO] 3. label: half track, probability: 0.0052872
[INFO] 4. label: restaurant, probability: 0.0040124
[INFO] 5. label: desktop computer, probability: 0.0033352
```

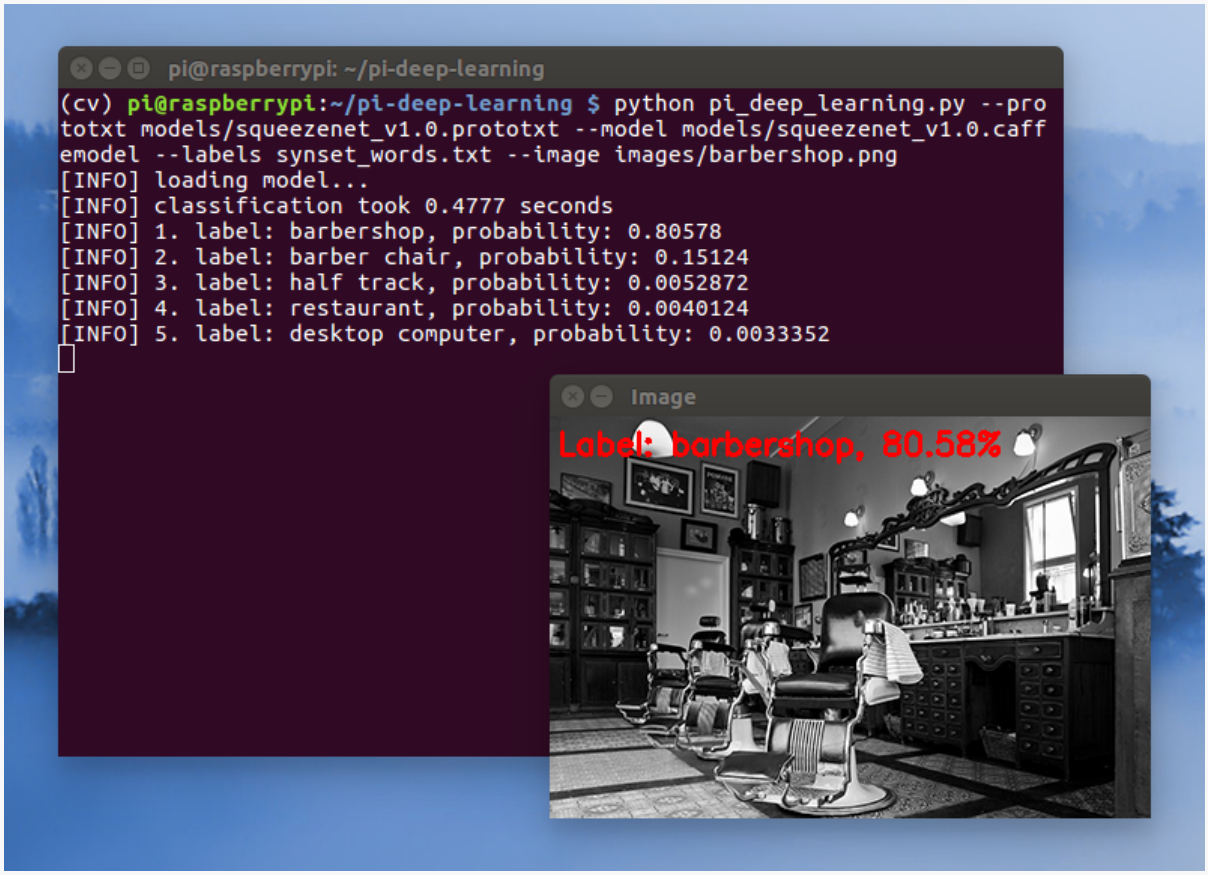


Figure 5: Squeezenet on the Raspberry Pi 3 also achieves performance gains using our optimized install of OpenCV 3.3.

Here we can see that SqueezeNet correctly classified the input image in **0.47 seconds**, another huge improvement from the 0.9 seconds from last week (47.78%).

Based on our results, it's *very* clear that our OpenCV optimizations have made a *significant* impact.

Summary

In today's blog post, you learned how to optimize your OpenCV install on the Raspberry Pi.

These optimizations came from updating our CMake command to include NEON and VFPV3.
When benchmarking OpenCV this leads to an approximate ***30% increase*** in speed. However, when applied strictly to the new `dnn` module in OpenCV 3, we are seeing an increase of over ***48%!***

I hope you enjoyed this tutorial and enjoy your optimized OpenCV + Raspberry Pi!

But before you go…

Be sure to check out other Raspberry Pi posts on my blog, and ***consider entering your email address in the form below*** **to be notified when future deep learning/Raspberry Pi posts are published here on PyImageSearch.**

Downloads:

If you would like to download the code and images used in this post, please enter your email address in the form below. Not only will you get a .zip of the code, I'll also send you a **FREE 11-page Resource Guide**on Computer Vision and Image Search Engines, including **exclusive techniques** that I don't post on this blog! Sound good? If so, enter your email address and I'll send you the code immediately!

Email address:

Your email address

DOWNLOAD THE CODE!

Resource Guide (it's totally free).

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

Your email address

DOWNLOAD THE GUIDE!

cnn, **convolutional neural network**, **deep learning**, **machine learning**, **neural nets**, **opencv**, **opencv 3**, **optimization**, **raspberry pi**

---

67 Responses to *Optimizing OpenCV on the Raspberry Pi*

Daniel October 9, 2017 at 10:30 am #

REPLY

Thanks for this great article!

One question: Does this also work on a Raspberry Pi 2?

Adrian Rosebrock October 9, 2017 at 12:11 pm #

REPLY

I haven't benchmarked or tested this procedure on the Raspberry Pi 2, but I imagine it will work there as well.

Steve Goldsmith October 9, 2017 at 11:21 am #

REPLY

I believe OpenCV since 3.2 will auto detect neon and vfpv for ARM, so there's no need to explicitly set it.

Adrian Rosebrock October 9, 2017 at 12:10 pm #

REPLY

The last time I compiled OpenCV on the Raspberry Pi it *did not* auto-detect NEON and VFPV. This was confirmed my manually inspecting my old OpenCV install and the benchmarks.

Denis Brion October 9, 2017 at 1:11 pm #

REPLY

Well, if swap partition is too small (cannot compile huge codes) and too large (eats SD space) and too fragile (wears out SD) , may be best solution (I did it with PCs) would be to :

(a) desactivate it "sudo swapoff -a" is likely to work (cannot check on a RPi fefore Friday), for the compiling session

(b) build another , bigger swap **file** on an external disk, (which should not be wearable)
Let us call $EXTDISK a path to an external disk:
dd if=/dev/zero of=${EXTDISK}/mySwap bs=1024 count=1000000
creates a huge , zero filled file on an external disk
sudo mkswap ${EXTDISK}/mySwap # format this file (it is not a partition)
sudo swapon ${EXTDISK}/mySwap # uses it as swap for the compiling session

(adapted
from https://www.centos.org/docs/5/html/5.2/Deployment_Guide/s2-swap-creating-file.html : it would be very unwise to put a swap file on a SD root , therefore $EXTFILE should be a path to a portion of a removable, infinite write disk)

Adrian Rosebrock October 9, 2017 at 4:52 pm #

REPLY

Great point regarding the external disk. Thank you for sharing, Denis 🙂

Mansoor October 9, 2017 at 1:25 pm #

REPLY

When you said in last week's post that you are going to show us optimization of opencv, I halted all operations and told all my students to wait for Adrian to show us how. And now i can gladly say that you are one of the best teacher we'll ever have! Thank you for everything! (Truly!)

Adrian Rosebrock October 9, 2017 at 4:51 pm #

REPLY

Thank you for the comment, Mansoor! Comments like these really put a smile on my face 🙂

Andrew October 9, 2017 at 1:33 pm #

REPLY

Fantastic! Do you think this build will work on a Pi Zero? Obviously, the Pi Z won't be as fast, but my current project needs the smaller footprint for a wearable. Thanks for all you do!

Adrian Rosebrock October 9, 2017 at 4:50 pm #

REPLY

I haven't tried on the Raspberry Pi Zero. If your goal is to work with real-time video I would not recommend the Pi Zero. With only a single core/processor thread it's just not fast enough.

rich October 18, 2017 at 11:14 am #

REPLY

I tried it on pi zero Ww, althouhg it works(bearly) it slowness could drive you to pull all your hair out! 🙂

Gary October 9, 2017 at 1:50 pm #

REPLY

Is it possible to use this with live video ?

Adrian Rosebrock October 9, 2017 at 4:49 pm #

REPLY

Yes, you just need to access your camera, whether that's a USB camera or Raspberry Pi camera module.

Denis Brion October 9, 2017 at 2:20 pm #

REPLY

"You can read more about large swap sizes corrupting memory cards on this page."

Better solution to avoid SD card corruption is to remove the swap file on the SD and use a swap file on an external disk – some cannot be corrupted …. – (for the compiling session, or for huge images…)
This is fully discussed in https://raspberrypi.stackexchange.com/questions/70/how-to-set-up-swap-space (and the traditional solution of dd + mkswap + swapon has been used for years in the PC world)

Adrian Rosebrock October 9, 2017 at 4:49 pm #

REPLY

Great point, thanks for sharing Denis!

memeka October 9, 2017 at 2:23 pm #

REPLY

You can manually edit the cmake scripts and replace the VFPV3 CFC flags with VFPV4, as rpi supports it. Might give you a bit of an extra speed..

Adrian Rosebrock October 9, 2017 at 4:48 pm #

REPLY

Thanks for sharing! I'll have to give this a try.

Flávio Rodrigues October 9, 2017 at 9:16 pm #

REPLY

Hi, Dr. Adrian. There is also a SqueezeNet v1.1 pre-trained model (2.4x less computation than v1.0, without sacrificing accuracy – https://github.com/DeepScale/SqueezeNet). I haven't tested it yet, but maybe it should also be worth a trial.

Adrian Rosebrock October 10, 2017 at 7:01 am #

REPLY

Hi Flávio — indeed, there is SqueezeNet v1.1; however, there is not a “deploy.prototxt” included in the repo for SqueezeNet v1.1 but there is one included for SqueezeNet v1.0. I decided to use SqueezeNet v1.0 to keep the explanation simple.

Thomas October 9, 2017 at 2:47 pm #

REPLY

Dear Adrian,

There is one point in your instructions/screenshots I didn’t quite understand.
The difference between building OpenCV for Python 2 or 3.
In the two screenshots from cmake, they show the path to your virtualenv python environment and to the ‘system’-installed-python and vice versa.

But the line below “Python (for build)” shows python 2.7 in both screenshots.

So in the end, what is used during the build process?
Is there a way to point to python 3.x – and does it make any difference?

Hope you can shed some light on this.
regards,
Thomas

Adrian Rosebrock October 9, 2017 at 4:48 pm #

REPLY

The “Python (for build)” statement in CMake is buggy. It will almost always report “Python 2.7” for the build. Instead, check the “Modules to be built” section of CMake and ensure either “python2” or “python3” appears in the output.

Thomas October 9, 2017 at 6:00 pm #

REPLY

Thank you for your quick answer!
Thomas

Aladdin Odeh October 9, 2017 at 3:48 pm #

REPLY

Thanks for this great tutorial and everything is work perfectly, I have a small question. is there a way to track the object after classify it , for example by draw a borders around it

Adrian Rosebrock October 9, 2017 at 4:47 pm #

REPLY

Are you referring to drawing the bounding box? If so, you can use the “cv2.rectangle” function. An example can be seen here.

Aladdin Odeh October 10, 2017 at 12:39 am #

REPLY

This is exactly what I want. 🙂 Thanks, Dr.Rosebrock

Santhosh M October 10, 2017 at 1:52 am #

REPLY

Thanks Adrian, very nice article.

My Pi 3 reports armv7 instead of armv8 (reports 32 bit instead of 64 bit). This has been discussed at various forums online and looks like the support for 64 bit is not yet available officially. openSUSE seems to have made an unofficial 64 bit images for the Pi 3. I’ve not tried it yet.

I’m not aware of the internals of OpenCV, its algorithms etc.

Are there any 64 bit arithmetic being used internally in OpenCV?
Is it possible to get better performance using the 64 bit capabilities of the Pi 3?
What is the best way to make use of the 64 bit capabilities (apart from the openSUSE version)?

Thank you.

Adrian Rosebrock October 10, 2017 at 6:56 am #

REPLY

OpenCV 64-bit bindings are used on most platforms where 64-bit is supported. While the Raspberry Pi 3 supports 64-bit the problem is actually with the operating system. There is no Raspbian-based operating system that currently supports 64-bit. Raspbian Stretch is still 32-bit.

There are various hacks that I’ve seen with people who use NOOBS who are able to flash a 64-bit version, but I’ll admit that this is not my

area of expertise.

REPLY

Santhosh M October 10, 2017 at 2:21 am #

Adrian, I also see the following from cmake/OpenCVCompilerOptions.cmake

if(ENABLE_VFPV3 AND NOT ENABLE_NEON)
add_extra_compiler_option(“-mfpu=vfpv3”)
endif()

Looks like if NEON is enabled vfpv3 is not enabled. Not sure why.

REPLY

memeka October 10, 2017 at 11:04 am #

you can manually add in the cmake file “-mfpu=neon-vfpv4” instead of neon, and you get best optimization for rpi3.

REPLY

Denis Brion October 10, 2017 at 3:07 am #

There is something which puzzles me, and I was surprised one could do some training with a RPi (it was a pleasant surprise, thanks for your great explanations) :

can one build opencv3 “only” for C++?
can one , in this option, compile and lik, either for ocv2 or for ocv3 (I hav some working programs written for opencv2; it is more comfortable, if one wants to port them to cv3, to have the old version still working, and compare it with the new version)

I am aware python is more comfortable and easier to read than plain c++ (and IMO space/time consumption differences can be neglected when one has to deal with arrays and images) , but I cannot learn at the same time c++ (coming from the Arduino world, I decided to begin with it) and python….

REPLY

Adrian Rosebrock October 10, 2017 at 6:58 am #

Yes, you can compile OpenCV 3 with only the C++ bindings and no Python, Java, etc. bindings. Simply turn off the Python bindings in your CMake configuration:

```
-D BUILD_opencv_python2=OFF \
-D BUILD_opencv_python3=OFF \
```

Anish jain October 10, 2017 at 3:50 am #

REPLY

Hi Adrian, thanks for another great tutorial!
I've been following your posts since long and I have a question
Will using NEON and VFPV3 for optimization cause any kind of side effects or decreased performance for the Raspberrypi in the long run, due to the extra load exerted on the processor for optimization?

Adrian Rosebrock October 10, 2017 at 6:59 am #

REPLY

That depends. If your system is always under heavy load of course it will lower than lifetime of the machine. But if your machine was under heavy load before the optimizations you would still be decreasing the lifetime of it. In short, I wouldn't worry about it.

Morgan October 10, 2017 at 4:09 am #

REPLY

Nice tutorial Adrian, very clear. Interesting to see your conclusions. I have spend way too much time over the last months trying to optimize opencv on the Pi 3.
To really squeeze everything out of opencv on the Pi, maybe it's time to go to the 64bit version, distributed by OpenSuse ?
[https://en.opensuse.org/HCL:Raspberry_Pi3]
Can I interest you in your next adventure, indoor location using ArCodes.
Have you looked into the new aruco library in OpenCV 3 ?
There is a stand-alone version based on opencv 2 (http://www.uco.es/investiga/grupos/ava/node/26) with python wrappers (https://github.com/fehlfarbe/python-aruco). But I think the new opencv aruco support is faster.
Either way, thanks and keep the tutorials coming.

Adrian Rosebrock October 10, 2017 at 7:00 am #

REPLY

Thanks for the comment, Morgan. I haven't used or tried ArUco yet, but I'll keep it in mind for a future tutorial.

Aladdin October 12, 2017 at 1:45 am #

REPLY

Thanks Adrian
I am wondering if I can use GoogLeNet instead of MobileNet for – Object detection with deep learning and OpenCV –

Adrian Rosebrock October 13, 2017 at 8:49 am #

REPLY

You can't swap in arbitrary deep neural networks to perform object detection. The neural network needs to be trained specifically to perform object detection using an object detection framework such as SSD, R-CNN, YOLO, etc.

Aladdin October 15, 2017 at 10:22 am #

REPLY

Do you know any pre-trained neural network with one of those OD frameworks that can detect more than those 20 classes [person, dog, cat …etc]?

Adrian Rosebrock October 16, 2017 at 12:25 pm #

REPLY

Take a look at the Caffe model zoo for pre-trained networks in Caffe format.

jsmith October 12, 2017 at 12:24 pm #

REPLY

Hi Adrian,

I got the following message while running both pi_deep_learning.py:

WARNING **: Error retrieving accessibility bus address: org.freedesktop.DBus.Error.ServiceUnknown: The name org.a11y.Bus was not provided by any .service files

But fixed by:

sudo apt install at-spi2-core

as per instructions here:

https://github.com/NixOS/nixpkgs/issues/16327

Thanks for the great tutorials!

Adrian Rosebrock October 13, 2017 at 8:40 am #

REPLY

Thanks for sharing!

OMH October 12, 2017 at 4:07 pm #

REPLY

Hi, great posts!

Unfortunately I have not been able to reproduce your results on the Raspberry PI3 😕 Running your example code the best I got was just above 0.7 seconds using squeezenet. Maybe something went wrong with my compilation of OpenCV, I don`t know.

However I have also followed your recipe for the PI3 but using an Asus Tinkerpad (kind of a Raspberry PI3 clone with more RAM and higher clock frequency but almost non existing support 😕), there I was far more successful! With the googlenet I achieved classification times around 0.41 seconds and with squeezenet around 0.25 seconds.

So future is looking good for my hobby project robot head now planning to use 3 fisheye cams for 360 degrees coverage plus two HD cams for stereo vision and distributed processing among multiple PI3s and Tinkerpads.

Looking forward to your future posts.

Adrian Rosebrock October 13, 2017 at 8:37 am #

REPLY

Congrats on the success with the Asus Tinkerpad! That's great. As for the Raspberry Pi, I'm not sure what may have happened. I would examine the CMake log and ensure that the NEON and VFPV3 optimizations were included in the compile.

DGS October 18, 2017 at 12:02 pm #

REPLY

Hi,
Could you please describe the process that you followed to install optimised OpenCV on Thinker board ?.I pretty much followed the same process mentioned here and do see similar classification time to yours, but I see the examples crashing on thinker board. what is the version of your Debian OS ?

Regards,
DGS

Hasanur Rashid October 14, 2017 at 1:50 pm #

REPLY

Hi Adrian,
I saw your great tutorial and then tried to cross compile opencv3 on my pc.After several attempts, I managed to build it with python2 and 3 support. Check out the gist if you are interested.
https://gist.github.com/hrshovon/70612b719bfda0becde46f0b9d2dfa36

Bruno October 14, 2017 at 5:01 pm #

REPLY

Hi Adrian
I followed your steps and finally success squeezenet model on virtual environment.
But, I didn't make any improvement compared with no virtual env.(Same processing time)

And I finally realized reason that NEON & VFPV3 were not enabled.
Here is Error message
CMAKE ERROR at
cmake/OpenCVCompilerOptimizations.cmake:399(message):
Required baseline optimization is not supported : VFPV3, NEON

How can I fix this?
Thanks

Adrian Rosebrock October 16, 2017 at 12:32 pm #

REPLY

Hi Bruno — which model Raspberry Pi are you using?

rich October 17, 2017 at 6:07 pm #

REPLY

Thank you very much Adrian, after nth times and numerous hours of trying, I finally get opencv compiled and installed on my py! Yippy. 🙂 Also this latest article is much better organized, the section about dependencies is much clearer, reduced a lot of human errors such as missing a line
and the swap file allows the make -j4 to use all 4 cores to compile, much faster then just 1 core.
Thank you again for spending the time writing such a detailed guild!

Adrian Rosebrock October 19, 2017 at 4:58 pm #

REPLY

Congrats on getting OpenCV compiled on your system, nice job!

Nanda October 17, 2017 at 9:38 pm #

REPLY

Hi, thanks for the great article. I get a lot of “compiler internal error” with “make -j4”, however there were no errors with “make -j1” but took 2 hours. Anything I am missing?

Adrian Rosebrock October 19, 2017 at 4:58 pm #

REPLY

It’s hard to say without seeing your log, but I would suggest proceeding with your `make -j1` command if OpenCV compiled without error for you.

Noble Koshy October 18, 2017 at 8:22 pm #

REPLY

Hi Adrian,
I installed OpenCV 3.3 according to your guide here: https://www.pyimagesearch.com/2017/09/04/raspbian-stretch-install-opencv-3-python-on-your-raspberry-pi/

Now I would like to install OpenCV using the NEON and FVPV3 optimizations mentioned here How would I first uninstall my current OpenCV install?

RPi 3 Model B
Raspbian Stretch

Noble Koshy October 18, 2017 at 8:22 pm #

REPLY

I unfortunately did not use a virtual env

Adrian Rosebrock October 19, 2017 at 4:44 pm #

REPLY