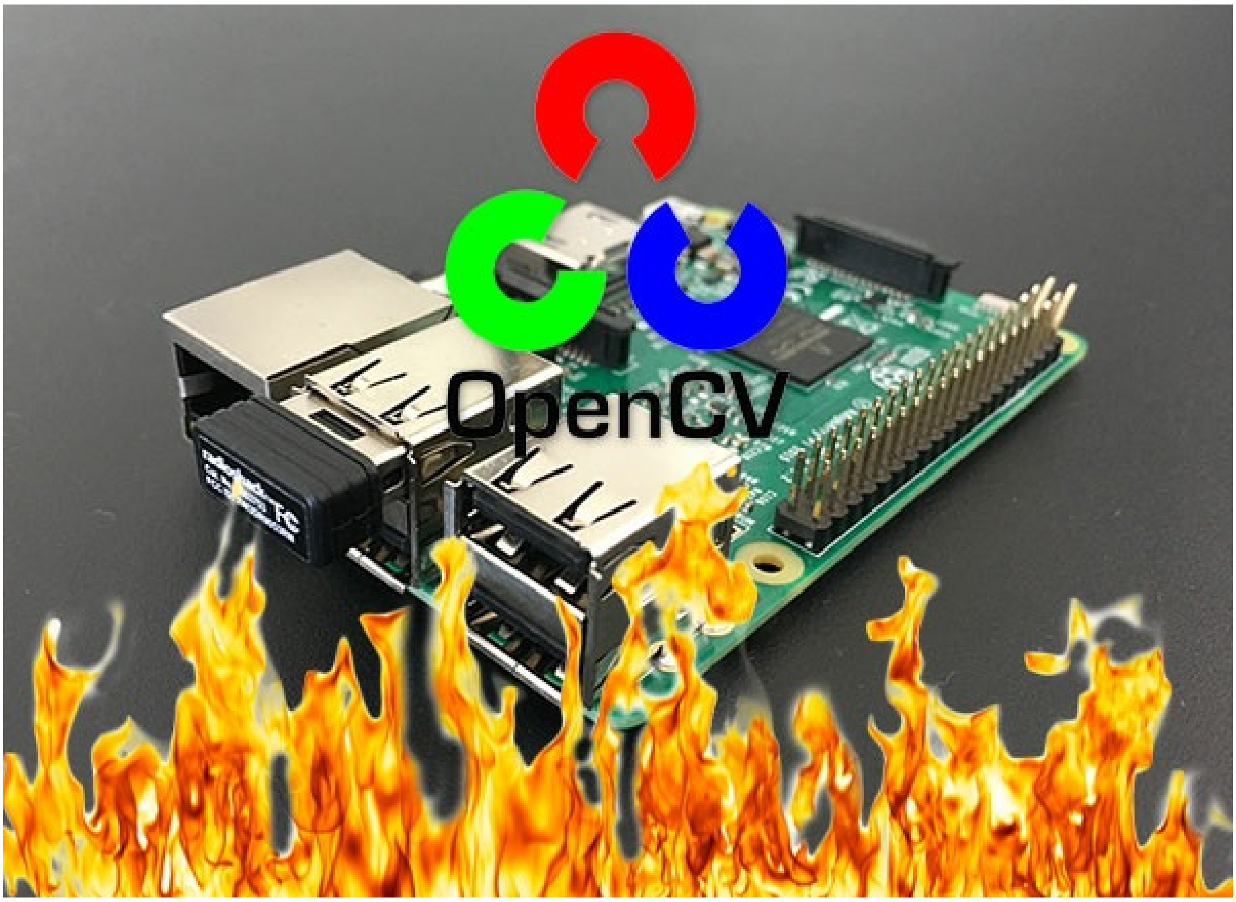
Optimizing OpenCV on the Raspberry Pi

pyimagesearch.com/2017/10/09/optimizing-opencv-on-the-raspberry-pi

Adrian Rosebrock

October 9, 2017



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 →

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
$ sudo raspi-config
And then reboot your Raspberry Pi:

Optimizing OpenCV on the Raspberry Pi
$ 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
$ sudo apt-get purge wolfram-engine
$ sudo apt-get purge libreoffice*
$ sudo apt-get clean
$ 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
$ 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 libpng12-dev
$ sudo apt-get install libavcodec-dev libavformat-dev libswscale-dev libv4l-dev
$ 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
$ cd ~
$ wget -O opencv.zip https://github.com/opencv/opencv/archive/3.4.7.zip
$ unzip opencv.zip
$ wget -O opencv_contrib.zip
https://github.com/opencv/opencv_contrib/archive/3.4.7.zip
$ 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 3.4.7**; however, as newer versions of OpenCV are released you can update the corresponding version numbers (**Note:** some screenshots show 3.3.0 as they were collected at the time when that was the highest version available).

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

```
$ 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
~/.bashrc
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

```
# virtualenv and virtualenvwrapper
export WORKON_HOME=$HOME/.virtualenvs
export VIRTUALENVWRAPPER_PYTHON=/usr/bin/python3
source /usr/local/bin/virtualenvwrapper.sh
```

From there, reload your

~/.bashrc
file to apply the changes to your current bash session:

Optimizing OpenCV on the Raspberry Pi

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

You'll need to run

source ~/.bashrc

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
$ 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
$ 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
$ workon cv
And from there configure your build:

Optimizing OpenCV on the Raspberry Pi
$ cd ~/opencv-3.4.7/
$ mkdir build
$ cd build
$ cmake -D CMAKE_BUILD_TYPE=RELEASE \
-D CMAKE_INSTALL_PREFIX=/usr/local \
-D OPENCV_EXTRA_MODULES_PATH=~/opencv_contrib-3.4.7/modules \
-D ENABLE_NEON=ON \

-D ENABLE_VFPV3=ON \
-D BUILD_TESTS=OFF \
-D INSTALL_PYTHON_EXAMPLES=OFF \
-D OPENCV_ENABLE_NONFREE=ON \
-D CMAKE_SHARED_LINKER_FLAGS='-latomic' \
-D BUILD_EXAMPLES=OFF ..
Notice how the NEON and VFPV3 flags have been enabled. These lines are highlighted.

Additionally, I've highlighted the NonFree algorithms flag (giving you the full install) as well as a special linker flag you need for OpenCV 3.4.7.

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

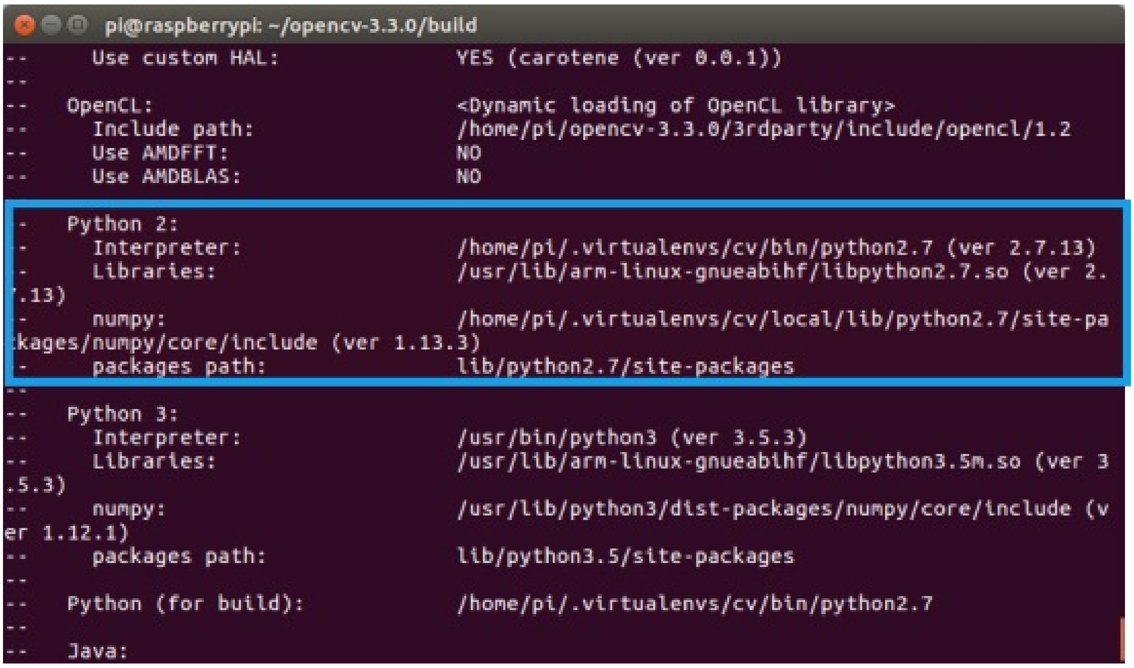


Figure 1: Running CMake to generate the build files for OpenCV 3.3. OpenCV will correctly be built with Python 2.7 and NumPy from our `cv` virtualenv.

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

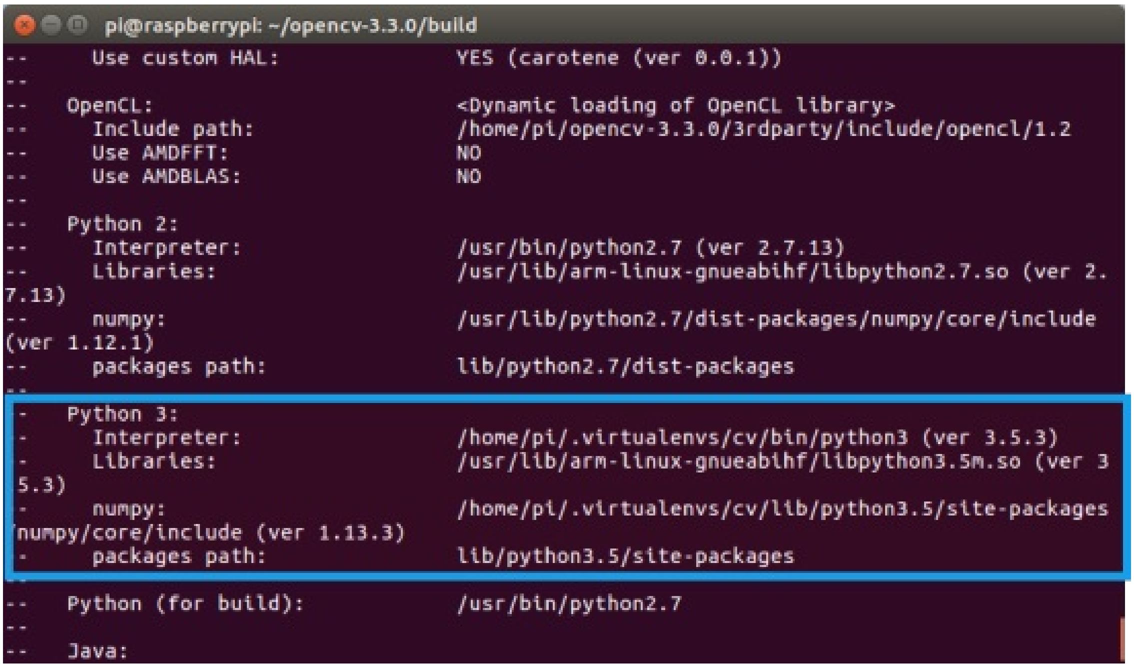


Figure 2: After running CMake, Python 3 + NumPy are correctly set from within our `cv` virtualenv on the Raspberry Pi.

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

```
# set size to absolute value, leaving empty (default) then uses computed value
# you most likely don't want this, unless you have an special disk situation
# 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
$ 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
$ make -j4

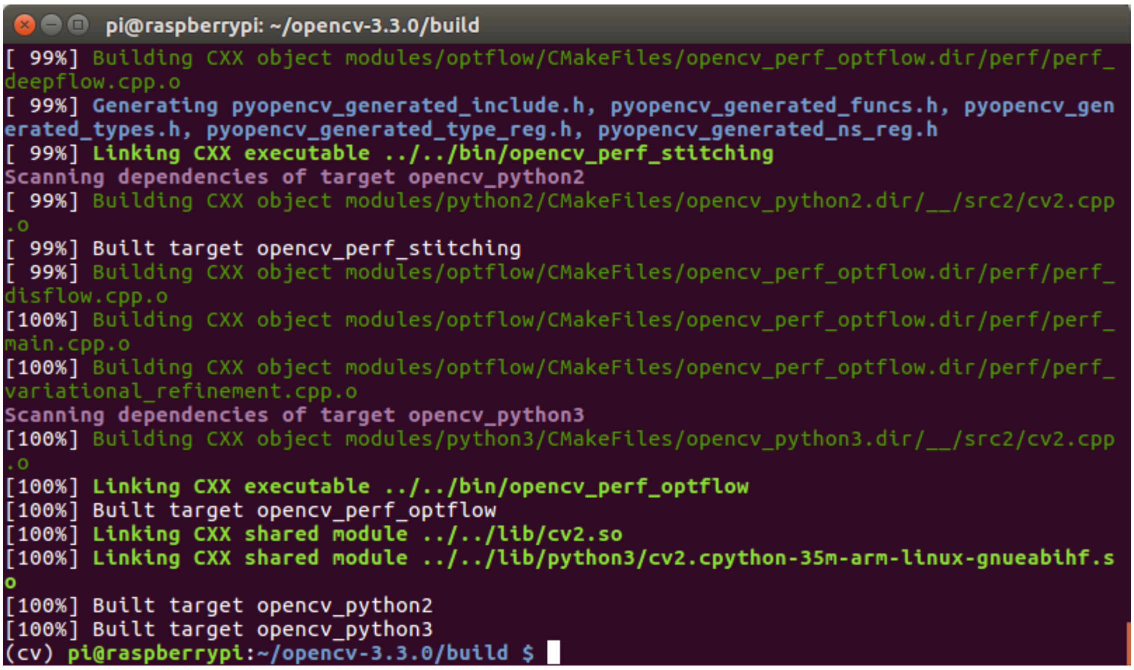


Figure 3: Our optimized compile of OpenCV 3.3 for the Raspberry Pi 3 has been completed successfully.

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
$ 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

```
$ 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

```
$ 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

```
$ source ~/.profile
$ workon cv
$ python
```

```
>>> import cv2
>>> cv2.__version__
'3.4.7'
>>>
```

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

```
$ python pi_deep_learning.py --prototxt models/bvlc_googlenet.prototxt \
--model models/bvlc_googlenet.caffemodel --labels synset_words.txt \
--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
```

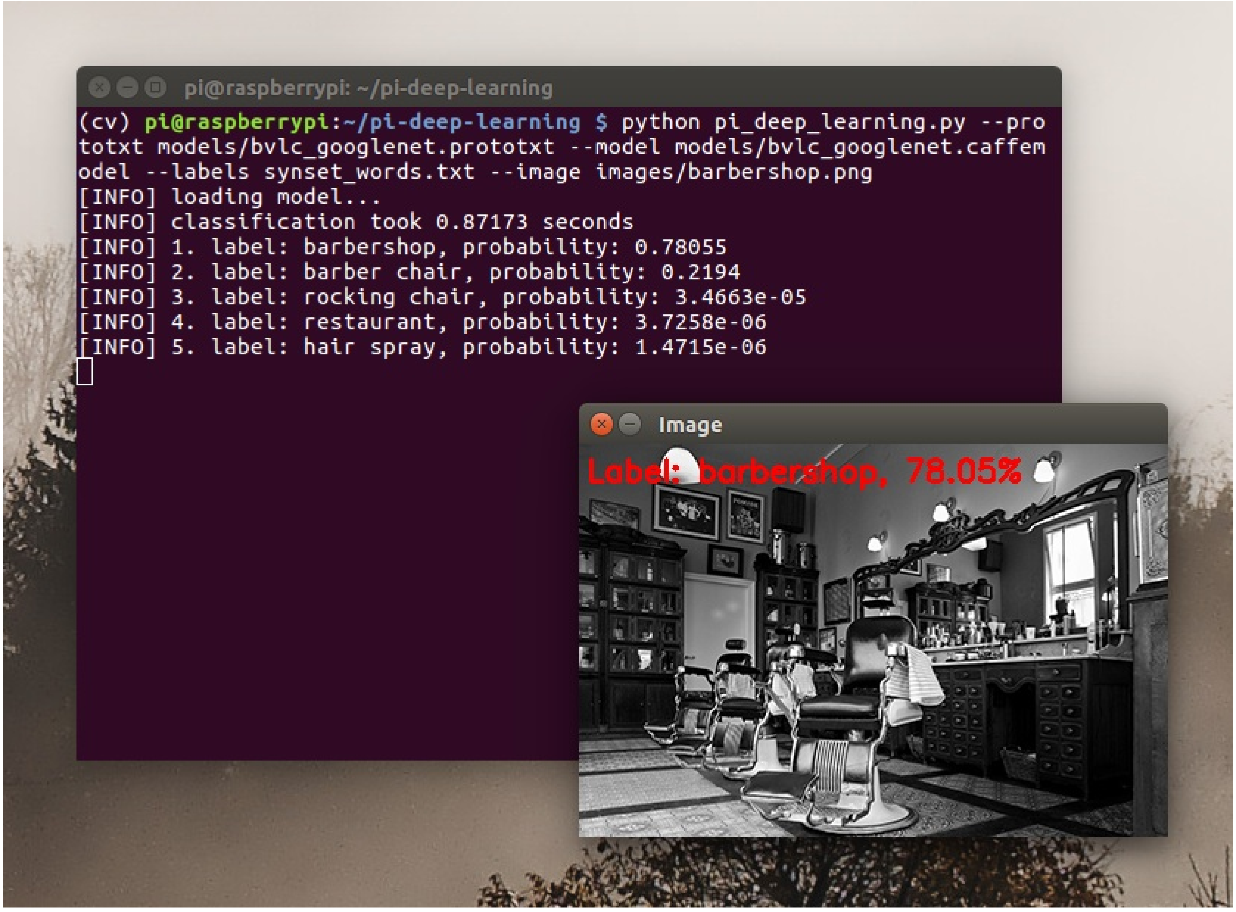


Figure 4: Running an image of a *"barbershop"* through GoogLeNet on the Raspberry Pi 3 with an optimized install of OpenCV 3.3 achieves a 48.82% speedup.

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

```
$ python pi_deep_learning.py --prototxt models/squeezenet_v1.0.prototxt \
--model models/squeezenet_v1.0.caffemodel --labels synset_words.txt \
--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
```

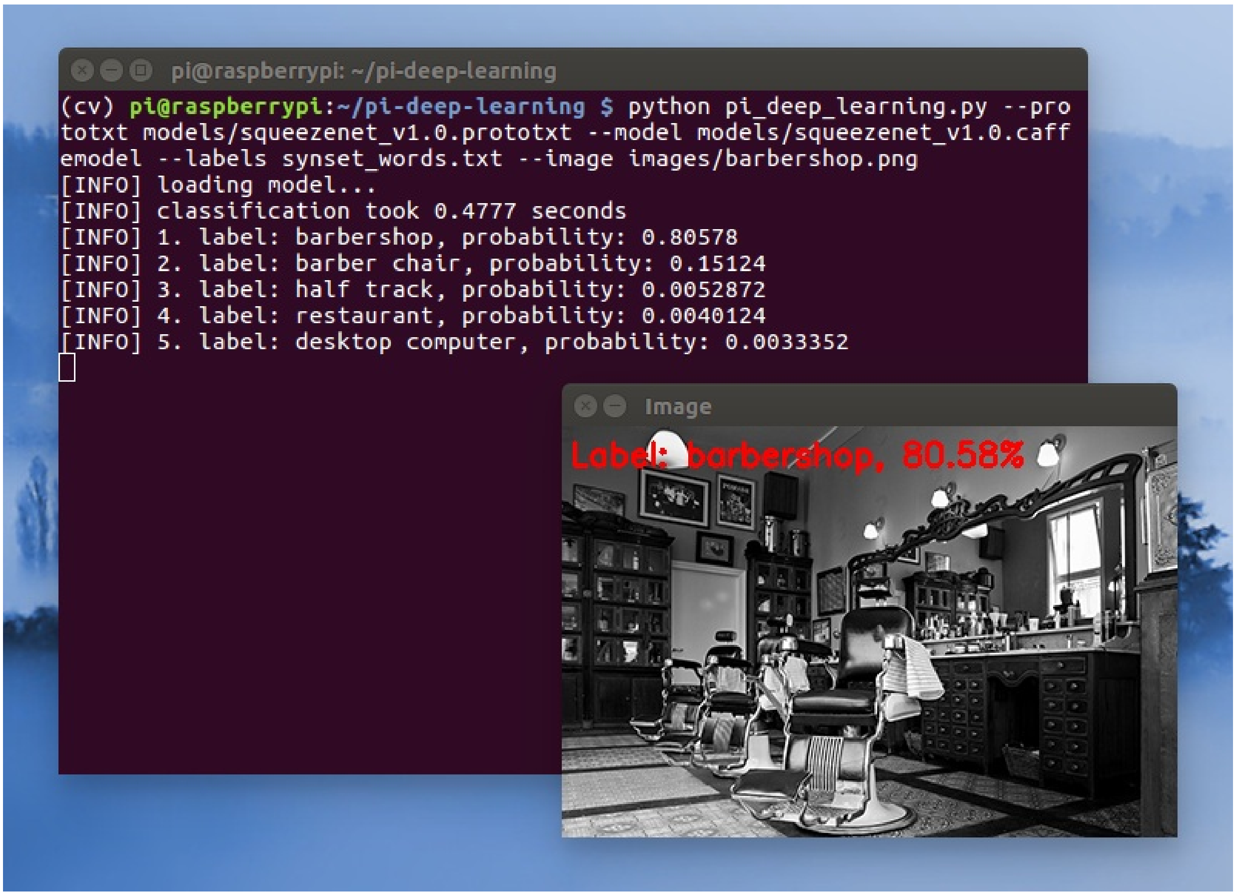


Figure 5: Squeezenet on the Raspberry Pi 3 also achieves performance gains using our optimized install of OpenCV 3.3.

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

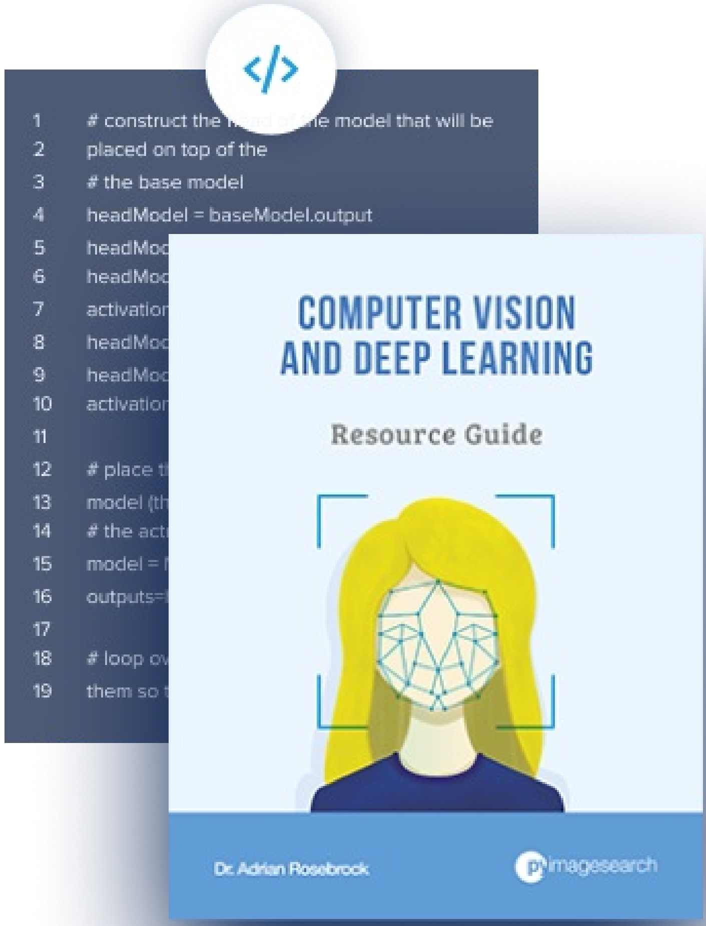
dnn

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.**



Download the Source Code and FREE 17-page Resource Guide

Enter your email address below to get a .zip of the code and a **FREE 17-page Resource Guide on Computer Vision, OpenCV, and Deep Learning.** Inside you'll find my hand-picked tutorials, books, courses, and libraries to help you master CV and DL!