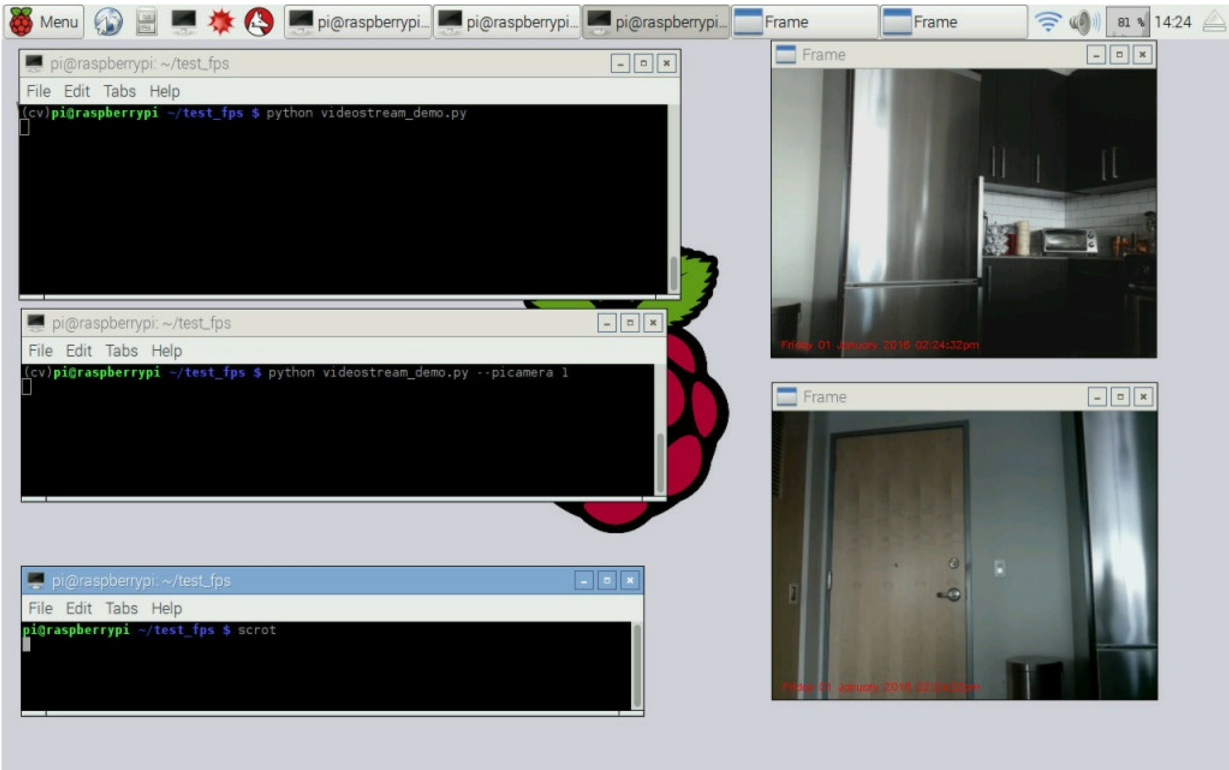
Unifying picamera and cv2.VideoCapture into a single class with OpenCV

pyimagesearch.com/2016/01/04/unifying-picamera-and-cv2-videocapture-into-a-single-class-with-opencv

Adrian Rosebrock January 4, 2016



Over the past two weeks on the PyImageSearch blog, we have discussed how to use threading to *increase our FPS processing rate* on ***both*** built-in/USB webcams, along with the Raspberry Pi camera module.

By utilizing threading, we learned that we can substantially reduce the affects of I/O latency, leaving the main thread to run without being blocked as it waits for I/O operations to complete (i.e., the reading of the most recent frame from the camera sensor).

Using this threading model, *we can dramatically increase our frame processing rate by upwards of 200%*.

While this increase in FPS processing rate is fantastic, there is still a (somewhat unrelated) problem that has been bothering me for for quite awhile.

You see, there are many times on the PyImageSearch blog where I write posts that are intended for use with a built-in or USB webcam, such as:

All of these posts rely on the

cv2.VideoCapture

method.

However, this reliance on

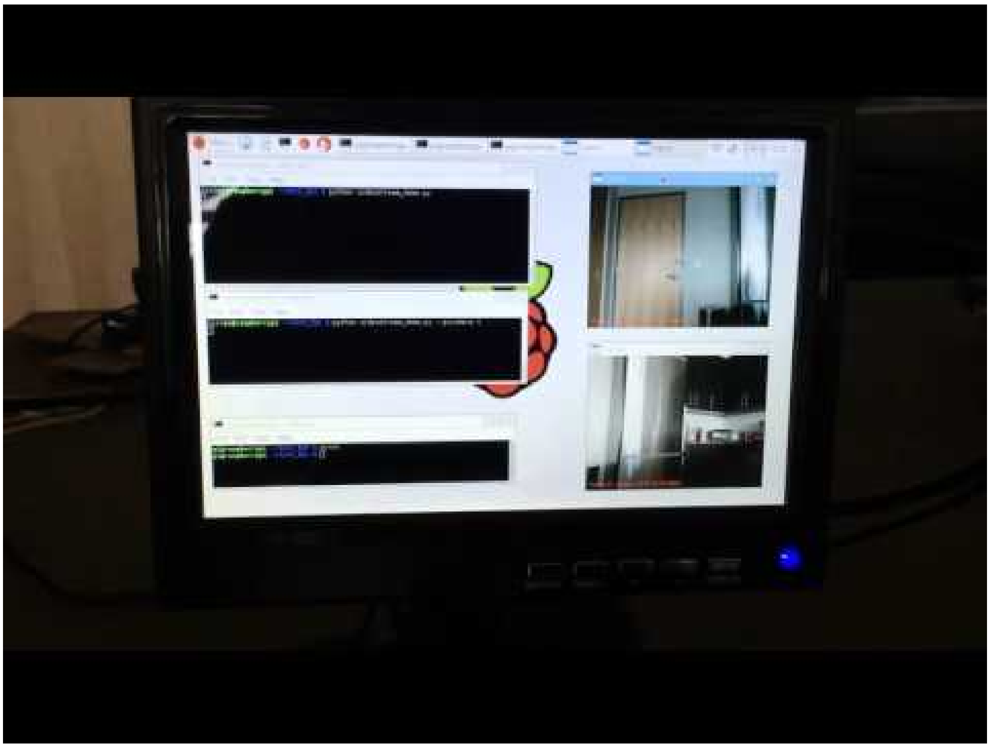
cv2.VideoCapture
 becomes a problem if you want to use the code on our Raspberry Pi. Provided that you are *not* using a USB camera with the Pi and *are in fact using the picamera module*, you'll need to modify the code to be compatible with
picamera
, as discussed in the accessing the Raspberry Pi Camera with Python and OpenCV post. While there are only a few required changes to the code (i.e., instantiating the

PiCamera
 class and swapping out the frame read loop), it can still be troublesome, especially if you are just getting started with Python and OpenCV.
Conversely, there are other posts on the PyImageSearch blog which use the

picamera
 module *instead of*
cv2.VideoCapture
. A great example of such a post is **home surveillance and motion detection with the Raspberry Pi, Python, OpenCV and Dropbox**. If you do not own a Raspberry Pi (or want to use a built-in or USB webcam instead of the Raspberry Pi camera module), you would again have to swap out a few lines of code.
Thus, the goal of this post is to a construct a ***unified interface*** to both

picamera
 and
cv2.VideoCapture
 using only a ***single class*** named
VideoStream
. This class will call either
WebcamVideoStream
 or
PiVideoStream
 based on the arguments supplied to the constructor.
Most importantly, our implementation of the

VideoStream
 class will allow future video processing posts on the PyImageSearch blog to run on either a **built-in webcam**, a **USB camera**, or the **Raspberry Pi camera module** — ***all without changing a single line of code!***
Read on to find out more.



Watch Video At: https://youtu.be/--90GjhAFH4

Looking for the source code to this post?

Jump Right To The Downloads Section →

If you recall from two weeks ago, we have already defined our threaded

WebcamVideoStream
 class for built-in/USB webcam access. And last week we defined the
PiVideoStream
 class for use with the Raspberry Pi camera module and the
picamera
 Python package.

Today we are going to unify these two classes into a single class named

VideoStream
 .

Depending on the parameters supplied to the

VideoStream
 constructor, the appropriate video stream class (either for the USB camera or
picamera

module) will be instantiated. This implementation of
VideoStream
will allow us to use the same set of code for all future video processing examples on the PyImageSearch blog.
Readers such as yourselves will only need to supply a *single command line argument* (or JSON configuration, etc.) to indicate whether they want to use their USB camera or the Raspberry Pi camera module — ***the code itself will not have to change one bit!***

As I've mentioned in the previous two blog posts in this series, the functionality detailed here is already implemented inside the imutils package.

If you do not have

imutils
already installed on your system, just use
pip
to install it for you:
Unifying picamera and cv2.VideoCapture into a single class with OpenCV

```
$ pip install imutils
```

Otherwise, you can upgrade to the latest version using:

Unifying picamera and cv2.VideoCapture into a single class with OpenCV

```
$ pip install --upgrade imutils
```

Let's go ahead and get started by defining the

VideoStream
class:
Unifying picamera and cv2.VideoCapture into a single class with OpenCV

```
# import the necessary packages
from webcamvideostream import WebcamVideoStream
class VideoStream:
def __init__(self, src=0, usePiCamera=False, resolution=(320, 240),
framerate=32):
# check to see if the picamera module should be used
if usePiCamera:
# only import the picamera packages unless we are
# explicity told to do so -- this helps remove the
# requirement of `picamera[array]` from desktops or
# laptops that still want to use the `imutils` package
from pivideostream import PiVideoStream
# initialize the picamera stream and allow the camera
# sensor to warmup
self.stream = PiVideoStream(resolution=resolution,
framerate=framerate)
# otherwise, we are using OpenCV so initialize the webcam
```

# stream
else:
self.stream = WebcamVideoStream(src=src)
On **Line 2** we import our

WebcamVideoStream
class that we use for accessing built-in/USB web cameras.
Line 5 defines the constructor to our

VideoStream
. The
src
keyword argument is only for the
cv2.VideoCapture
function (abstracted away by the
WebcamVideoStream
class), while
usePiCamera
,
resolution
, and
framerate
are for the
picamera
module.
We want to take special care to *not make any assumptions* about the the type of hardware or the Python packages installed by the end user. If a user is programming on a laptop or a desktop, then it's *extremely unlikely* that they will have the

picamera
module installed.
Thus, we'll only import the

PiVideoStream
class (which then imports dependencies from
picamera
) if the
usePiCamera
boolean indicator is ***explicitly defined*** (**Lines 8-18**).
Otherwise, we'll simply instantiate the

WebcamVideoStream
(**Lines 22 and 23**) which requires no dependencies other than a working OpenCV installation.
Let's define the remainder of the

VideoStream
class:
Unifying picamera and cv2.VideoCapture into a single class with OpenCV

```
def start(self):
# start the threaded video stream
return self.stream.start()
def update(self):
# grab the next frame from the stream
self.stream.update()
def read(self):
# return the current frame
return self.stream.read()
def stop(self):
# stop the thread and release any resources
self.stream.stop()
```

As we can see, the

start
,
update
,
read
, and
stop
methods simply call the corresponding methods of the
stream
which was instantiated in the constructor.
Now that we have defined the

VideoStream
class, let’s put it to work in our
videostream_demo.py
driver script:
Unifying picamera and cv2.VideoCapture into a single class with OpenCV

```
# import the necessary packages
from imutils.video import VideoStream
import datetime
import argparse
import imutils
import time
import cv2
# construct the argument parse and parse the arguments
ap = argparse.ArgumentParser()
ap.add_argument("-p", "--picamera", type=int, default=-1,
help="whether or not the Raspberry Pi camera should be used" )
```

```
args = vars(ap.parse_args())
# initialize the video stream and allow the cammera sensor to warmup
vs = VideoStream(usePiCamera=args["picamera"] > 0).start()
time.sleep(2.0)
```

We start off by importing our required Python packages (**Lines 2-7**) and parsing our command line arguments (**Lines 10-13**). We only need a single switch here,

--picamera

, which is used to indicate whether the Raspberry Pi camera module or the built-in/USB webcam should be used. We'll default to the built-in/USB webcam.

Lines 16 and 17 instantiate our

VideoStream

and allow the camera sensor to warmup.

At this point, all the hard work is done! We simply need to start looping over frames from the camera sensor:

Unifying picamera and cv2.VideoCapture into a single class with OpenCV

```
# loop over the frames from the video stream
while True:
# grab the frame from the threaded video stream and resize it
# to have a maximum width of 400 pixels
frame = vs.read()
frame = imutils.resize(frame, width=400)
# draw the timestamp on the frame
timestamp = datetime.datetime.now()
ts = timestamp.strftime("%A %d %B %Y %I:%M:%S%p")
cv2.putText(frame, ts, (10, frame.shape[0] - 10), cv2.FONT_HERSHEY_SIMPLEX,
0.35, (0, 0, 255), 1)
# show the frame
cv2.imshow("Frame", frame)
key = cv2.waitKey(1) & 0xFF
# if the `q` key was pressed, break from the loop
if key == ord("q"):
break
# do a bit of cleanup
cv2.destroyAllWindows()
vs.stop()
```

On **Line 20** we start an infinite loop that continues until we press the

q

key.

Line 23 calls the

read

method of

VideoStream
 which returns the most recently read
frame
 from the stream (again, either a USB webcam stream or the Raspberry Pi camera module).
We then resize the frame (**Line 24**), draw the current timestamp on it (**Lines 27-30**), and finally display the frame to our screen (**Lines 33 and 34**).

This is obviously a trivial example of a video processing pipeline, but keep in mind the goal of this post is to simply demonstrate how we can create a unified interface to both the

picamera
 module and the
cv2.VideoCapture
 function.

Testing out our unified interface

To test out our

VideoStream
 class, I used:

- A Raspberry Pi 2 with *both* a Raspberry Pi camera module and a USB camera (a Logitech C920 which is plug-and-play compatible with the Pi).
- My OSX laptop with built-in webcam.

To access the built-in camera on my OSX machine, I executed the following command:

Unifying picamera and cv2.VideoCapture into a single class with OpenCV
```
$ python videostream_demo.py
```

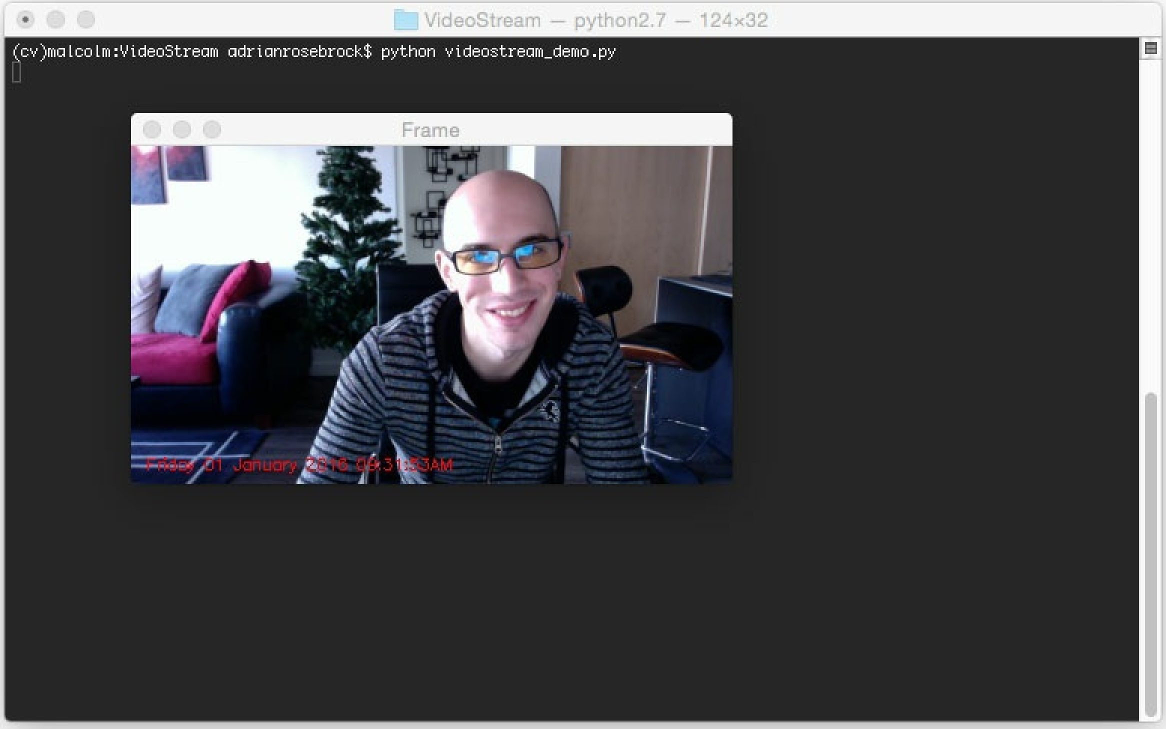


Figure 1: Accessing the built-in camera on my OSX machine with Python and OpenCV.

As you can see, frames are read from my webcam and displayed to my screen.

I then moved over to my Raspberry Pi where I executed the same command to access the USB camera:

Unifying picamera and cv2.VideoCapture into a single class with OpenCV

```
$ python videostream_demo.py
```

Followed by this command to read frames from the Raspberry Pi camera module:

Unifying picamera and cv2.VideoCapture into a single class with OpenCV

```
$ python videostream_demo.py --picamera 1
```

The results of executing these commands in two separate terminals can be seen below:

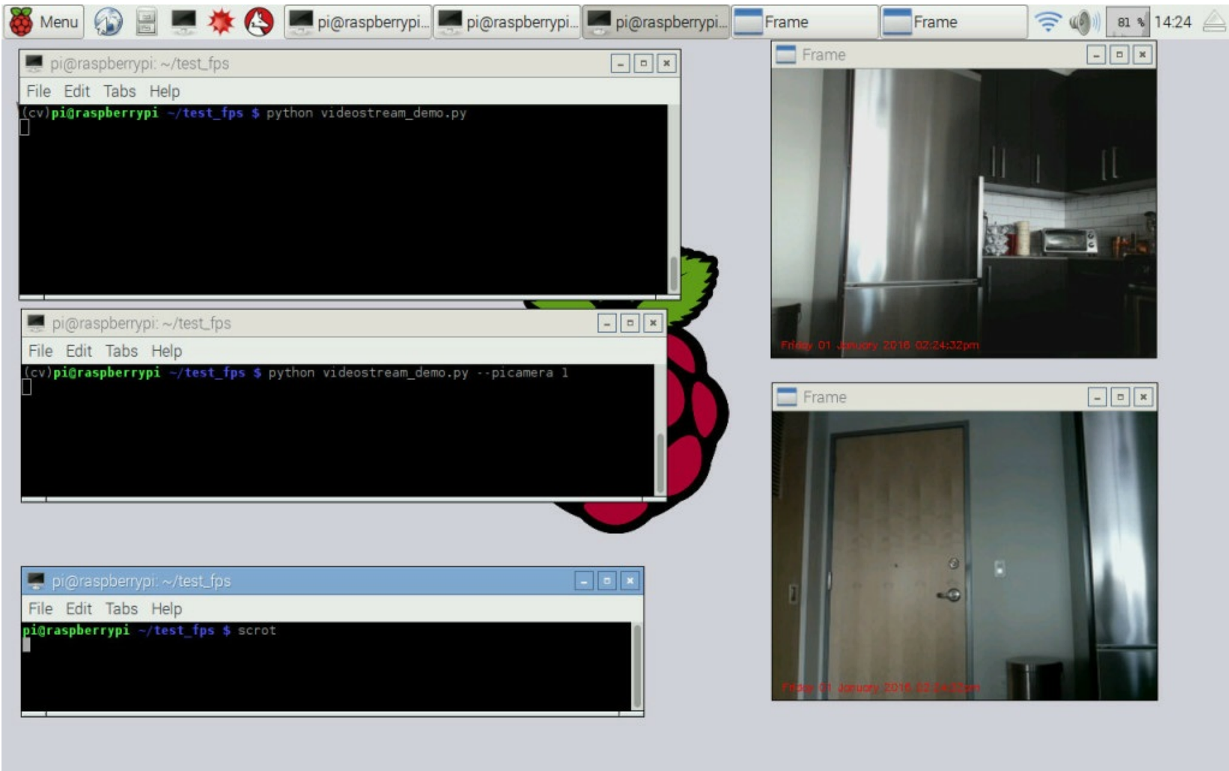


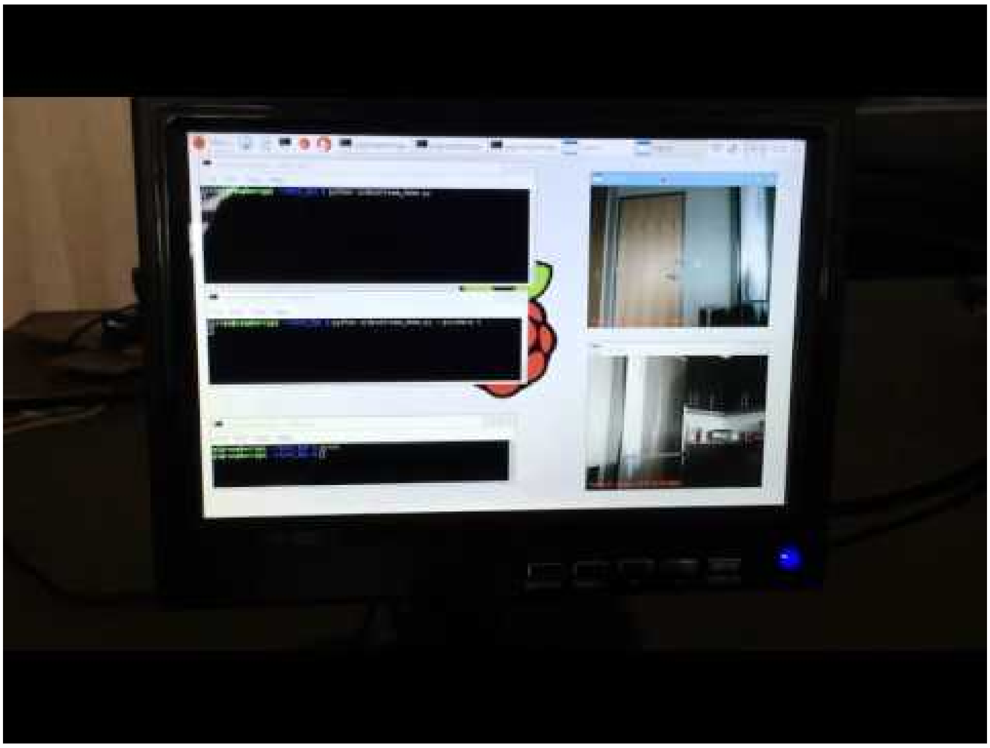
Figure 2: Accessing both the Raspberry Pi camera module and a USB camera on my Raspberry Pi using the exact same Python class.

As you can see, *the only thing that has changed* is the command line arguments where I supply

--picamera 1

, indicating that I want to use the Raspberry Pi camera module — ***not a single line of code needed to be modified!***

You can see a video demo of both the USB camera and the Raspberry Pi camera module being used simultaneously below:



Watch Video At: https://youtu.be/--90GjhAFH4

Summary

This blog post was the third and final installment in our series on increasing FPS processing rate and decreasing I/O latency on both USB cameras and the Raspberry Pi camera module.

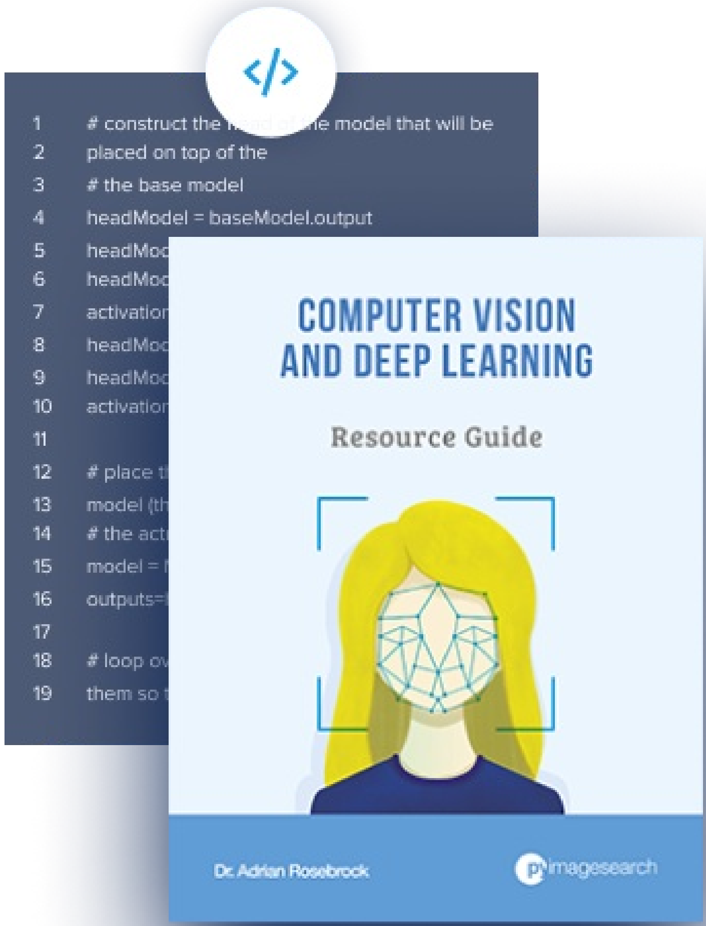
We took our implementations of the (threaded)

WebcamVideoStream
 and
PiVideoStream
 classes and unified them into a single
VideoStream
 class, allowing us to seamlessly access *either* built-in/USB cameras or the Raspberry Pi camera module.
This allows us to construct Python scripts that will run on ***both*** laptop/desktop machines along with the the Raspberry Pi *without having to modify a single line of code* — provided that we supply some sort of method to indicate which camera we would like to use, of course, This can easily be accomplished using command line arguments, JSON configuration files, etc.

In future blog posts where video processing is performed, I'll be using the

VideoStream
 class to make the code examples compatible with both your USB camera and the Raspberry Pi camera module — *no longer will you have to adjust the code based on your setup!*
Anyway, I hope you enjoyed this series of posts. If you found me doing a *series* of blog posts (rather than *one-off* posts on a specific topic) beneficial, please let me know in the comments thread.

And also consider signing up for the PyImageSearch Newsletter using the form below to be notified when new blog posts are published!



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!