Raspberry Pi: Deep learning object detection with OpenCV

by **Adrian Rosebrock** on October 16, 2017 in **Deep Learning**, **OpenCV 3**, **Raspberry Pi**

A few weeks ago I demonstrated how to perform real-time object detection using deep learning and OpenCV on a standard laptop/desktop.

After the post was published I received a number of emails from PyImageSearch readers who were curious if the Raspberry Pi could *also*be used for real-time object detection.

The short answer is “kind of”…

…but only if you set your expectations accordingly.

Even when applying our optimized OpenCV + Raspberry Pi install the Pi is only capable of **getting up to ~0.9 frames per second** when applying deep learning for object detection with Python and OpenCV.

Is that fast enough?

Well, that depends on your application.

If you’re attempting to detect objects that are quickly moving through your field of view, likely not.

But if you’re monitoring a low traffic environment with slower moving objects, the Raspberry Pi could indeed be fast enough.

In the remainder of today’s blog post we’ll be reviewing two methods to perform deep learning-based object detection on the Raspberry Pi.

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

Raspberry Pi: Deep learning object detection with OpenCV

Today’s blog post is broken down into two parts.

In the first part, we’ll benchmark the Raspberry Pi for real-time object detection using OpenCV and Python. This benchmark will come from the exact code we used for our laptop/desktop deep learning object detectorfrom a few weeks ago.

I’ll then demonstrate how to use multiprocessing to create an alternate method to object detection using the Raspberry Pi. This method may or may not be useful for your particular application, but at the very least it will

give you an idea on different methods to approach the problem.

Object detection and OpenCV benchmark on the Raspberry Pi

The code we'll discuss in this section is is identical to our previous post on *Real-time object detection with deep learning and OpenCV*; therefore, I will not be reviewing the code exhaustively.

For a deep dive into the code, please see the original post.

Instead, we'll simply be using this code to benchmark the Raspberry Pi for deep learning-based object detection.

To get started, open up a new file, name it `real_time_object_detection.py` , and insert the following code:

```
Raspberry Pi: Deep learning object detectio                    Python
1  # import the necessary packages
2  from imutils.video import VideoStream
3  from imutils.video import FPS
4  import numpy as np
5  import argparse
6  import imutils
7  import time
8  import cv2
```

We then need to parse our command line arguments:

```
Raspberry Pi: Deep learning object detectio                    Python
10 # construct the argument parse and parse the arguments
11 ap = argparse.ArgumentParser()
12 ap.add_argument("-p", "--prototxt", required=True,
13     help="path to Caffe 'deploy' prototxt file")
14 ap.add_argument("-m", "--model", required=True,
15     help="path to Caffe pre-trained model")
16 ap.add_argument("-c", "--confidence", type=float, default=0.2,
17     help="minimum probability to filter weak detections")
18 args = vars(ap.parse_args())
```

Followed by performing some initializations:

```
Raspberry Pi: Deep learning object detectio                    Python
20 # initialize the list of class labels MobileNet SSD was trained
21 # detect, then generate a set of bounding box colors for each cl
22 CLASSES = ["background", "aeroplane", "bicycle", "bird", "boat",
23     "bottle", "bus", "car", "cat", "chair", "cow", "diningtable"
24     "dog", "horse", "motorbike", "person", "pottedplant", "sheep
25     "sofa", "train", "tvmonitor"]
26 COLORS = np.random.uniform(0, 255, size=(len(CLASSES), 3))
27
28 # load our serialized model from disk
29 print("[INFO] loading model...")
30 net = cv2.dnn.readNetFromCaffe(args["prototxt"], args["model"])
```

We initialize `CLASSES` , our class labels, and corresponding `COLORS` , for on-frame text and bounding boxes (**Lines 22-26**), followed by loading the serialized neural network model (**Line 30**).

Next, we'll initialize the video stream object and frames per second counter:

```
Raspberry Pi: Deep learning object detectio                    Python
32 # initialize the video stream, allow the camera sensor to warm u
33 # and initialize the FPS counter
34 print("[INFO] starting video stream...")
35 vs = VideoStream(src=0).start()
36 # vs = VideoStream(usePiCamera=True).start()
37 time.sleep(2.0)
38 fps = FPS().start()
```

Wwe initialize the video stream and allow the camera warm up for 2.0 seconds (**Lines 35-37**).

On **Line 35** we initialize our `VideoStream` using a USB camera If you are using the Raspberry Pi camera module you'll want to comment out **Line 35** and uncomment **Line 36** (which will enable you to access the Raspberry Pi camera module via the `VideoStream` class).

From there we start our `fps` counter on **Line 38**.

We are now ready to loop over frames from our input video stream:

```
Raspberry Pi: Deep learning object detectio                    Python
40 # loop over the frames from the video stream
41 while True:
42     # grab the frame from the threaded video stream and resize i
43     # to have a maximum width of 400 pixels
44     frame = vs.read()
45     frame = imutils.resize(frame, width=400)
46
47     # grab the frame dimensions and convert it to a blob
48     (h, w) = frame.shape[:2]
49     blob = cv2.dnn.blobFromImage(cv2.resize(frame, (300, 300)),
50         0.007843, (300, 300), 127.5)
51
52     # pass the blob through the network and obtain the detection
53     # predictions
54     net.setInput(blob)
55     detections = net.forward()
```

Lines 41-55 simply grab and resize a `frame`, convert it to a `blob`, and pass the `blob` through the neural network, obtaining the `detections` and bounding box predictions.

From there we need to loop over the `detections` to see what objects were detected in the `frame`:

```
Raspberry Pi: Deep learning object detectio                    Python
57     # loop over the detections
58     for i in np.arange(0, detections.shape[2]):
59         # extract the confidence (i.e., probability) associated
60         # the prediction
61         confidence = detections[0, 0, i, 2]
62
63         # filter out weak detections by ensuring the `confidence
64         # greater than the minimum confidence
65         if confidence > args["confidence"]:
66             # extract the index of the class label from the
67             # `detections`, then compute the (x, y)-coordinates
68             # the bounding box for the object
69             idx = int(detections[0, 0, i, 1])
70             box = detections[0, 0, i, 3:7] * np.array([w, h, w,
71             (startX, startY, endX, endY) = box.astype("int")
72
```

```
            # draw the prediction on the frame
            label = "{}: {:.2f}%".format(CLASSES[idx],
                confidence * 100)
            cv2.rectangle(frame, (startX, startY), (endX, endY),
                COLORS[idx], 2)
            y = startY - 15 if startY - 15 > 15 else startY + 15
            cv2.putText(frame, label, (startX, y),
                cv2.FONT_HERSHEY_SIMPLEX, 0.5, COLORS[idx], 2)
```

On **Lines 58-80**, we loop over our `detections` . For each detection we examine the `confidence` and ensure the corresponding probability of the detection is above a predefined threshold. If it is, then we extract the class label and compute *(x ,y)* bounding box coordinates. These coordinates will enable us to draw a bounding box around the object in the image along with the associated class label.

From there we'll finish out the loop and do some cleanup:

Raspberry Pi: Deep learning object detectio Python

```
    # show the output 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

    # update the FPS counter
    fps.update()

# stop the timer and display FPS information
fps.stop()
print("[INFO] elapsed time: {:.2f}".format(fps.elapsed()))
print("[INFO] approx. FPS: {:.2f}".format(fps.fps()))

# do a bit of cleanup
cv2.destroyAllWindows()
vs.stop()
```

Lines 82-91 close out the loop — we show each frame, `break` if 'q' key is pressed, and update our `fps` counter.

The final terminal message output and cleanup is handled on **Lines 94-100**.

Now that our brief explanation of `real_time_object_detection.py` is finished, let's examine the results of this approach to obtain a baseline.

Go ahead and use the ***"Downloads"*** section of this post to download the source code and pre-trained models.

From there, execute the following command:

Raspberry Pi: Deep learning object detection Shell

```
$ python real_time_object_detection.py \
    --prototxt MobileNetSSD_deploy.prototxt.txt \
    --model MobileNetSSD_deploy.caffemodel
[INFO] loading model...
[INFO] starting video stream...
[INFO] elapsed time: 54.70
```

```
[INFO] approx. FPS: 0.90
```

As you can see from my results we are obtaining ~0.9 frames per second throughput using this method and the Raspberry Pi.

Compared to the 6-7 frames per second using our laptop/desktop we can see that the Raspberry Pi is *substantially* slower.

That's not to say that the Raspberry Pi is unusable when applying deep learning object detection, but you need to set your expectations on what's realistic (even when applying our OpenCV + Raspberry Pi optimizations).

***Note:** For what it's worth, I could only obtain **0.49 FPS** when **NOT** using our optimized OpenCV + Raspberry Pi install — that just goes to show you how much of a difference NEON and VFPV3 can make.*

A different approach to object detection on the Raspberry Pi

Using the example from the previous section we see that calling `net.forward()` is a ***blocking operation*** — the rest of the code in the `while` loop is not allowed to complete until `net.forward()` returns the `detections` .

So, what if `net.forward()` was not a blocking operation?

Would we able to obtain a faster frames per second throughput?

Well, that's a loaded question.

No matter what, it will take approximately a little over a second for `net.forward()` to complete using the Raspberry Pi and this particular architecture — that cannot change.

But what we *can* do is create a separate process that is solely responsible for applying the deep learning object detector, thereby unblocking the main thread of execution and allow our `while` loop to continue.

Moving the predictions to separate process will give the *illusion* that our Raspberry Pi object detector is running faster than it actually is, when in reality the `net.forward()` computation is still taking a little over one second.

The only problem here is that our output object detection predictions will lag behind what is currently being displayed on our screen. If you detecting *fast-moving objects* you may miss the detection entirely, or at the very least, the object will be out of the frame before you obtain your detections from the neural network.

Therefore, this approach should only be used for *slow-moving objects*where we can tolerate lag.

To see how this multiprocessing method works, open up a new file, name it `pi_object_detection.py` , and insert the following code:

Raspberry Pi: Deep learning object detectio Python

```
# import the necessary packages
```

```
2  from imutils.video import VideoStream
3  from imutils.video import FPS
4  from multiprocessing import Process
5  from multiprocessing import Queue
6  import numpy as np
7  import argparse
8  import imutils
9  import time
10 import cv2
```

For the code walkthrough in this section, I'll be pointing out and explaining the *differences* (there are quite a few) compared to our non-multprocessing method.

Our imports on **Lines 2-10** are mostly the same, but notice the imports of `Process` and `Queue` from Python's multiprocessing package.

Next, I'd like to draw your attention to a new function, `classify_frame` :

```
Raspberry Pi: Deep learning object detectio                    Python
12 def classify_frame(net, inputQueue, outputQueue):
13     # keep looping
14     while True:
15         # check to see if there is a frame in our input queue
16         if not inputQueue.empty():
17             # grab the frame from the input queue, resize it, an
18             # construct a blob from it
19             frame = inputQueue.get()
20             frame = cv2.resize(frame, (300, 300))
21             blob = cv2.dnn.blobFromImage(frame, 0.007843,
22                 (300, 300), 127.5)
23
24             # set the blob as input to our deep learning object
25             # detector and obtain the detections
26             net.setInput(blob)
27             detections = net.forward()
28
29             # write the detections to the output queue
30             outputQueue.put(detections)
```

Our new `classify_frame` function is responsible for our multiprocessing — later on we'll set it up to run in a child process.

The `classify_frame` function takes three parameters:

- `net` : the neural network object.
- `inputQueue` : our FIFO (first in first out) queue of frames for object detection.
- `outputQueue` : our FIFO queue of detections which will be processed in the main thread.

This child process will loop continuously until the parent exits and effectively terminates the child.

In the loop, if the `inputQueue` contains a `frame` , we grab it, and then pre-process it and create a `blob` (**Lines 16-22**), just as we have done in the previous script.

From there, we send the `blob` through the neural network (**Lines 26-27**) and place the `detections` in an `outputQueue` for processing by the parent.

Now let's parse our command line arguments:

```
Raspberry Pi: Deep learning object detectio                    Python
32 # construct the argument parse and parse the arguments
33 ap = argparse.ArgumentParser()
34 ap.add_argument("-p", "--prototxt", required=True,
35     help="path to Caffe 'deploy' prototxt file")
36 ap.add_argument("-m", "--model", required=True,
37     help="path to Caffe pre-trained model")
38 ap.add_argument("-c", "--confidence", type=float, default=0.2,
39     help="minimum probability to filter weak detections")
40 args = vars(ap.parse_args())
```

There is no difference here — we are simply parsing the same command line arguments on **Lines 33-40**.

Next we initialize some variables just as in our previous script:

```
Raspberry Pi: Deep learning object detectio                    Python
42 # initialize the list of class labels MobileNet SSD was trained
43 # detect, then generate a set of bounding box colors for each cl
44 CLASSES = ["background", "aeroplane", "bicycle", "bird", "boat",
45     "bottle", "bus", "car", "cat", "chair", "cow", "diningtable"
46     "dog", "horse", "motorbike", "person", "pottedplant", "sheep
47     "sofa", "train", "tvmonitor"]
48 COLORS = np.random.uniform(0, 255, size=(len(CLASSES), 3))
49
50 # load our serialized model from disk
51 print("[INFO] loading model...")
52 net = cv2.dnn.readNetFromCaffe(args["prototxt"], args["model"])
```

This code is the same — we initialize class labels, colors, and load our model.

Here's where things get different:

```
Raspberry Pi: Deep learning object detectio                    Python
54 # initialize the input queue (frames), output queue (detections)
55 # and the list of actual detections returned by the child proces
56 inputQueue = Queue(maxsize=1)
57 outputQueue = Queue(maxsize=1)
58 detections = None
```

On **Lines 56-58** we initialize an `inputQueue` of frames, an `outputQueue` of detections, and a `detections` list.

Our `inputQueue` will be populated by the parent and processed by the child — it is the input to the child process. Our `outputQueue` will be populated by the child, and processed by the parent — it is output from the child process. Both of these queues trivially have a size of *one* as our neural network will only be applying object detections to one frame at a time.

Let's initialize and start the child process:

```
Raspberry Pi: Deep learning object detectio                    Python
60 # construct a child process *indepedent* from our main process o
61 # execution
62 print("[INFO] starting process...")
63 p = Process(target=classify_frame, args=(net, inputQueue,
64     outputQueue,))
```

```
65 p.daemon = True
66 p.start()
```

It is very easy to construct a child process with Python's multiprocessing module — simply specify the `target` function and `args` to the function as we have done on **Lines 63 and 64**.

Line 65 specifies that `p` is a daemon process, and **Line 66** kicks the process off.

From there we'll see some more familiar code:

```
Raspberry Pi: Deep learning object detectio                    Python
68 # initialize the video stream, allow the cammera sensor to warmu
69 # and initialize the FPS counter
70 print("[INFO] starting video stream...")
71 vs = VideoStream(src=0).start()
72 # vs = VideoStream(usePiCamera=True).start()
73 time.sleep(2.0)
74 fps = FPS().start()
```

Don't forget to change your video stream object to use the PiCamera if you desire by switching which line is commented (**Lines 71 and 72**).

Once our `vs` object and `fps` counters are initialized, we can loop over the video frames:

```
Raspberry Pi: Deep learning object detectio                    Python
76 # loop over the frames from the video stream
77 while True:
78     # grab the frame from the threaded video stream, resize it,
79     # grab its dimensions
80     frame = vs.read()
81     frame = imutils.resize(frame, width=400)
82     (fH, fW) = frame.shape[:2]
```

On **Lines 80-82**, we read a frame, resize it, and extract the width and height.

Next, we'll work our our queues into the flow:

```
Raspberry Pi: Deep learning object detectio                    Python
84     # if the input queue *is* empty, give the current frame to
85     # classify
86     if inputQueue.empty():
87         inputQueue.put(frame)
88
89     # if the output queue *is not* empty, grab the detections
90     if not outputQueue.empty():
91         detections = outputQueue.get()
```

First we check if the `inputQueue` is empty — if it is empty, we put a frame in the `inputQueue` for processing by the child (**Lines 86 and 87**). Remember, the child process is running in an infinite loop, so it will be processing the `inputQueue` in the background.

Then we check if the `outputQueue` is not empty — if it is not empty (something is in it), we grab the `detections` for processing here in the parent (**Lines 90 and 91**). When we call `get()` on the `outputQueue`, the detections are returned and the `outputQueue` is now momentarily empty.

If you are unfamiliar with Queues or if you want a refresher, see this documentation.

Let's process our detections:

Raspberry Pi: Deep learning object detectio Python

```
93      # check to see if our detectios are not None (and if so, we
94      # draw the detections on the frame)
95      if detections is not None:
96          # loop over the detections
97          for i in np.arange(0, detections.shape[2]):
98              # extract the confidence (i.e., probability) associ
99              # with the prediction
100             confidence = detections[0, 0, i, 2]
101
102             # filter out weak detections by ensuring the `confi
103             # is greater than the minimum confidence
104             if confidence < args["confidence"]:
105                 continue
106
107             # otherwise, extract the index of the class label f
108             # the `detections`, then compute the (x, y)-coordin
109             # of the bounding box for the object
110             idx = int(detections[0, 0, i, 1])
111             dims = np.array([fW, fH, fW, fH])
112             box = detections[0, 0, i, 3:7] * dims
113             (startX, startY, endX, endY) = box.astype("int")
114
115             # draw the prediction on the frame
116             label = "{}: {:.2f}%".format(CLASSES[idx],
117                 confidence * 100)
118             cv2.rectangle(frame, (startX, startY), (endX, endY)
119                 COLORS[idx], 2)
120             y = startY - 15 if startY - 15 > 15 else startY + 1
121             cv2.putText(frame, label, (startX, y),
122                 cv2.FONT_HERSHEY_SIMPLEX, 0.5, COLORS[idx], 2)
```

If our `detections` list is populated (it is not `None`), we loop over the detections as we have done in the previous section's code.

In the loop, we extract and check the `confidence` against the threshold (**Lines 100-105**), extract the class label index (**Line 110**), and draw a box and label on the frame (**Lines 111-122**).

From there in the while loop we'll complete a few remaining steps, followed by printing some statistics to the terminal, and performing cleanup:

Raspberry Pi: Deep learning object detectio Python

```
124     # show the output frame
125     cv2.imshow("Frame", frame)
126     key = cv2.waitKey(1) & 0xFF
127
128     # if the `q` key was pressed, break from the loop
129     if key == ord("q"):
130         break
131
132     # update the FPS counter
133     fps.update()
134
135 # stop the timer and display FPS information
136 fps.stop()
137 print("[INFO] elapsed time: {:.2f}".format(fps.elapsed()))
```

```
138 print("[INFO] approx. FPS: {:.2f}".format(fps.fps()))
139
140 # do a bit of cleanup
141 cv2.destroyAllWindows()
142 vs.stop()
```

In the remainder of the loop, we display the frame to the screen (**Line 125**) and capture a key press and check if it is the quit key at which point we break out of the loop (**Lines 126-130**). We also update our `fps` counter.

To finish out, we stop the `fps` counter, print our time/FPS statistics, and finally close windows and stop the video stream (**Lines 136-142**).

Now that we're done walking through our new multiprocessing code, let's compare the method to the single thread approach from the previous section.

Be sure to use the ***"Downloads"*** section of this blog post to download the source code + pre-trained MobileNet SSD neural network. From there, execute the following command:

```
Raspberry Pi: Deep learning object detection                Shell
1 $ python pi_object_detection.py \
2     --prototxt MobileNetSSD_deploy.prototxt.txt \
3     --model MobileNetSSD_deploy.caffemodel
4 [INFO] loading model...
5 [INFO] starting process...
6 [INFO] starting video stream...
7 [INFO] elapsed time: 48.55
8 [INFO] approx. FPS: 27.83
```

Here you can see that our `while` loop is capable of processing 27 frames per second. However, this throughput rate is an illusion — the neural network running in the background is still only capable of processing 0.9 frames per second.

Note: *I also tested this code on the Raspberry Pi camera module and was able to obtain 60.92 frames per second over 35 elapsed seconds.*

The difference here is that we can obtain real-time *throughput* by displaying each new input frame in real-time and then drawing any previous `detections` on the current frame.

Once we have a new set of `detections` we then draw the new ones on the frame.

This process repeats until we exit the script. **The downside is that we see substantial lag.** There are clips in the above video where we can see that all objects have clearly left the field of view…

…however, our script still reports the objects as being present.

Therefore, you should ***consider only using this approach when***:

1. Objects are slow moving and the previous detections can be used as an approximation to the new location.
2. Displaying the actual frames themselves in real-time is paramount to user experience.

Summary

In today's blog post we examined using the Raspberry Pi for object detection using deep learning, OpenCV, and Python.

As our results demonstrated we were able to get up to **0.9 frames per second**, which is not fast enough to constitute real-time detection. That said, given the limited processing power of the Pi, 0.9 frames per second is still reasonable for some applications.

We then wrapped up this blog post by examining an alternate method to deep learning object detection on the Raspberry Pi by using multiprocessing. Whether or not this second approach is suitable for you is again highly dependent on your application.

If your use case involves low traffic object detection where the objects are slow moving through the frame, then you can certainly consider using the Raspberry Pi for deep learning object detection. However, if you are developing an application that involves many objects that are fast moving, you should instead consider faster hardware.

Thanks for reading and enjoy!

And if you're interested in studying deep learning in more depth, be sure to take a look at my new book, *Deep Learning for Computer Vision with Python*. Whether this is the first time you've worked with machine learning and neural networks or you're already a seasoned deep learning practitioner, my new book is engineered from the ground up to help you reach expert status.

Just click here to start your journey to deep learning mastery.

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).

cnn, **coco**, **convolutional neural network**, **deep learning**, **machine learning**, **neural nets**, **opencv 3**, **optimization**, **raspberry pi**, **real-time**, **video**, **video stream**

41 Responses to *Raspberry Pi: Deep learning object detection with OpenCV*

Mr.Odeh October 16, 2017 at 11:16 am #

REPLY

Thanks, dr.adrian for this great article!
It is working fine with me, but I have a small question.
I want to detect insects in real time using raspberry, do you recommend any pre-trained module that can do the object detection for insects not just persons, dogs, sofa …etc? if there isn't what should u do in general to achieve my aim?
thanks again

Adrian Rosebrock October 16, 2017 at 12:14 pm #

REPLY

I'm not aware of a pre-trained model that specifically detects insects. I would suggest fine-tuning an existing, pre-trained neural network. I discussing fine-tuning inside Deep Learning for Computer Vision with Python.

JBeale October 16, 2017 at 11:55 am #

REPLY

Very impressive! My own experiments on RPi was about 3 or 4 seconds per frame. So almost 1 fps is quite an improvement from that. With a moderately wide-angle lens that could already be useful, unless you must have the object almost completely fill the frame. Does this code

fully utilize all 4 cores on the RPi 3, or is there potentially some additional parallelization possible?

Adrian Rosebrock October 16, 2017 at 12:13 pm #

REPLY

There are always more optimizations that can be made, it's just a matter of if it's worth it. The fully utilize all cores to their maximum potential we would need OpenCL (which to my knowledge) The Raspberry Pi does not support.

JBeale October 17, 2017 at 1:46 pm #

REPLY

Thanks. I have yet to try your code, but if (for example) it only uses 2 cores and it's CPU-bound rather than memory or I/O bound, you ought to get a speedup by simply instantiating two separate processes, one looking at odd frames and one doing even frames. But maybe it's not that easy; you might run out of memory.

JBeale October 18, 2017 at 12:14 pm #

REPLY

Speaking of OpenCL on Raspbery Pi: it is not 100% complete, but:
09-OCT-2017 : I present to you VC4CL (VideoCore IV OpenCL):
https://www.raspberrypi.org/forums/viewtopic.php?t=194952

Adrian Rosebrock October 18, 2017 at 3:55 pm #

REPLY

Awesome, thanks for sharing.

Jay October 27, 2017 at 1:53 am #

Came across this article by accident. Great site. I realise this is a python based site but what are the speed improvements were this to be implemented in C++ for comparison sake?

Adrian Rosebrock October 27, 2017 at 11:15 am #

Hi Jay — I'm glad you enjoyed the blog post. Python is just a wrapper around the original C/C++ code for OpenCV. So the speed will be very similar.

Dayle October 16, 2017 at 12:18 pm #

REPLY

Hi Adrian,

Thanks a ton for remembering us Pi enthusiasts. I first got interested in image analysis after someone stole your beer, but was afraid you would lose interest in the Pi after purchasing the beast.

Looking forward to diving in to this post and reading the new book.

Cheers,
Dayle

Adrian Rosebrock October 16, 2017 at 12:35 pm #

REPLY

Hi Dayle — I certainly have not lost interested in the Raspberry Pi, I've just primarily been focusing on deep learning tutorials lately 🙂

Anish October 16, 2017 at 2:25 pm #

REPLY

How is this method different from using Squezenet for object detection on a raspberry pi?
The one you posted a couple of weeks ago?
Also what are the pros and cons of using squezenet over this method?

Adrian Rosebrock October 17, 2017 at 9:37 am #

REPLY

SqueezeNet is an image classifier. It takes an entire image and returns a single class label. It does no object detection or localization.

The SSD and Faster R-CNN frameworks can be used for object detection. It requires that you take an architecture (SqueezeNet, VGGNet, etc.) and then train it using the object detection framework.

This will minimize the joint loss between class label prediction AND localization.

The gist is that vanilla SqueezeNet and SSD are two totally different frameworks.

If you're interested in learning more about deep learning (and how these architectures differ), I would definitely suggest working through Deep Learning for Computer Vision with Python where I cover these methods in detail.

Ahmad October 16, 2017 at 6:33 pm #

REPLY

i have this error , there is a problem here -> args = vars(ap.parse_args())

usage: real_time_object_detection.py [-h] -p PROTOTXT -m MODEL [-c CONFIDENCE]
real_time_object_detection.py: error: argument -p/–prototxt is required

Adrian Rosebrock October 17, 2017 at 9:34 am #

REPLY

Please read up on command line arguments.

Komoriii October 16, 2017 at 9:38 pm #

REPLY

Impressive tutorial.This article helped me a lot,thank you!

Adrian Rosebrock October 17, 2017 at 9:34 am #

REPLY

Fantastic, I'm glad to hear it Komoriii! 🙂

Sachin October 17, 2017 at 1:36 am #

REPLY

Great post as always, Adrian! I have learned a lot about computer vision from the content on your site.

Regarding doing AI on the Pi, I would personally not do detection and recognition on an edge device. At least not until they ship a Pi with an AI chip and a decent GPU! And maybe not even then, due to the high power (electricity) consumption of AI. I'd much rather use the Pi as a

sensor + basic signal processor, WiFi over all the video / sensor signals to a CPU box, and run all the algorithms on that box.
So I guess I agree with your conclusions.

Adrian Rosebrock October 17, 2017 at 9:33 am #

REPLY

Hi Sachin — thanks for the comment. I actually discuss the tradeoffs of using the Raspberry Pi for deep learning in this post. In general, I do agree with you that a Raspberry Pi should not be used for deep learning unless under very specific circumstances.

Abhishek October 17, 2017 at 3:35 am #

REPLY

Hi Adrian, i love ur work, Sir can you please tell me how i can compute :the (x, y)-coordinates of the bounding box for the object if i'm using Squeezenet instead of MobileNet SSD caffe Model on my raspberry pi 3…..what i supposed to change in "box = detections[0, 0, i, 3:7] * np.array([w, h, w, h])" so that it will work with detecting object in squeezenet with highest probablity (I'm able to find the index of object with highest probability till now with your previous post on deep learning) any help is appreciated 🙂 [i have raspberrian stretch with opencv3.30 -dev installed(neon optimized)]

Adrian Rosebrock October 17, 2017 at 9:31 am #

REPLY

Are you using your own custom trained SqueezeNet using the SSD framework? Keep in mind that you cannot swap out networks trained for image classification and use them for object detection.

Abhishek October 17, 2017 at 11:16 pm #

REPLY

Thanks for reply :), i figured that out eventually (Previously i was using SqueezeNet v1.1 imageclassifier instead of SSD framework) but i found another great SqueezeNet-SSD (based on Squeezenet v1.1)

https://github.com/chuanqi305/SqueezeNet-SSD

I benchmarked it using this script but see merely any difference from MobileNetSSD …..both have same FPS around 1.72FPS(opencv optimized) on normal script and

29.4FPS(opencv optimized) using this Multithreaded script…..Through Squeezenet v1.1 (around 0.4 ms on raspberry pi 3) is way faster than any other image classifier, why this Squeezenet-SSD is slower ? I'm totally confused :\

Adrian Rosebrock October 19, 2017 at 4:56 pm #

REPLY

SqueezeNet v1.1 is slower because it utilizes ResNet-like modules. These increase accuracy, but slow the network down a bit.

David Killen October 17, 2017 at 4:46 am #

REPLY

trivial point, no need to publish, but you say 'net.forwad()' vice 'net.forward()' at least twice

Adrian Rosebrock October 17, 2017 at 9:30 am #

REPLY

Thanks for letting me know, David! I have updated the post.

M.Komijani October 17, 2017 at 8:37 am #

REPLY

Hello Adrian,

Thanks for this great article!
Actually, I'm a starter in deep learning, but I want to use the Raspberry Pi for deep learning object detection.
One question: Does x-nor net improves the speed results?
(https://pjreddie.com/media/files/XNOR_ECCV2.pdf)

Adrian Rosebrock October 17, 2017 at 9:27 am #

REPLY

I haven't used XNOR net to benchmark it, but from the paper the argument is that you can use XNOR net to speedup the network. You end up saving (approximately) 32x memory and 58x faster convolutional operations.

Ying October 18, 2017 at 5:09 am #

REPLY

hi Adrian,

Can we only detect people or car (i.e. specific class) by changing the python code?

REPLY

Adrian Rosebrock October 19, 2017 at 4:53 pm #

Yes. Check the `idx` of the predicted class and filter out the ones you are uninterested in.

REPLY

jsmith October 18, 2017 at 11:27 am #

Hi Adrian,

I've been wanting to do this for months, and it was this that got me to your website, so thank you!

I have been trying to tweak the code so that I can grab the frame when an object is detected and save that as a .jpg in a folder as:

/Pictures/{label}/{label}_{confidence}.jpg

following the 'Accessing the Raspberry Pi Camera with OpenCV and Python' tutorial, so I can have my own dataset by using the Pi to do all the hard work.

However I keep getting an mmalError message.

How would you go about taking a frame from the Pi when it detects an object and saving that frame in a folder with that object's class so you can have a dataset to work with?

Thanks!

REPLY

Adrian Rosebrock October 19, 2017 at 4:51 pm #

I would suggest debugging this line-by-line. Try to determine what line is throwing the error by inserting "print" statements. If you can provide that, I can try to point you in the right direction.

From there, you can use the `cv2.imwrite` to save your image to disk. You can format your filename using the detected label and associated probability returned by `net.forward`.

Roald October 18, 2017 at 3:52 pm #

REPLY

Hi Adrian,

I'm having issues that net.forward() seems to return inconsistent results. For example, I have two frames. One frame with my cat, one frame without. If I process frame-without-cat, the cat is not found. If I process frame-with-cat, it is correctly found. However, if I do this:

detect frame-without-cat
detect frame-with-cat
detect frame-without-cat

I get the following results:
cat not detected
cat detected
cat detected

Which is inconsistent, as the third and first frame should have the same result. However, if for each detection I reload the model, this issue does not occur. It looks as though the net retains previous detections?

Do you have any idea what this could be? If need be, I can provide example data and source.

Adrian Rosebrock October 19, 2017 at 4:47 pm #

REPLY

Hi Roald — this is indeed strange; however, I would double-check your images and 100% verify that you are passing in the correct images as you expect. The network should not be retaining any type of memory from previous detections. Secondly, check the confidence (i.e., probability) of your false-positives and see if you can increase the confidence to filter out these weak detections.

Noble October 20, 2017 at 12:03 am #

REPLY

Hi Adrian,

Downloaded the example for this post and ran it:

$ $ python3 real_time_object_detection.py
usage: real_time_object_detection.py [-h] -p PROTOTXT -m MODEL [-c CONFIDENCE]
real_time_object_detection.py: error: the following arguments are required: -p/–prototxt, -m/–model

How do I specify the path for the protext file and the model file to ap.add_argument. They are all in the same folder.

REPLY

Adrian Rosebrock October 22, 2017 at 8:46 am #

Please read up on command line arguments. This will enable you to learn more about command line basics. Furthermore, I also present examples on how to run the Python script via the command line inside this blog post.

REPLY

Human October 20, 2017 at 5:15 pm #

i want to track a ball is that code reliable to do the task

REPLY

Adrian Rosebrock October 22, 2017 at 8:37 am #

I cover ball/object tracking inside this post.

REPLY

Apramey October 21, 2017 at 7:45 am #

Hello Adrian Sir,
I'm great fan of all your articles and I ought learn more from you.
I'm presently running on ubuntu mate on raspberry pi 3, I even optimized pi, the way you told in previous post. I removed the unnecessary applications of ubuntu mate which I'm not using. The code runs without any error. But the problem is GPU rendering, I get the frame, but I can't visualize the video it's recording. it continuously lags after code starts running.

REPLY

Adrian Rosebrock October 22, 2017 at 8:35 am #

The Raspberry Pi will only be able to process ~1 frame per second using this deep learning-based object detection method so the lag is entirely normal. Is there another type of lag you are referring to?

REPLY

Reed October 29, 2017 at 1:01 am #

Hi Adrian
I tried to run the codes above, but the result was
$ python pi_object_detection.py –prototxt
MobileNetSSD_deploy.prototxt.txt –model
MobileNetSSD_deploy.caffemodel
[INFO] loading model…
[INFO] starting process…
[INFO] starting video stream…
…
(h, w) = image.shape[:2]
AttributeError: 'NoneType' object has no attribute 'shape'

and I also read the post on 26th Dec 2016, still have no clue
what should I do?

Adrian Rosebrock October 30, 2017 at 3:19 pm #

Hi Reed — this error usually occurs when the webcam didn't properly grab an image. You could put the following between lines 80 and 81:

```
if frame is None:
continue
```