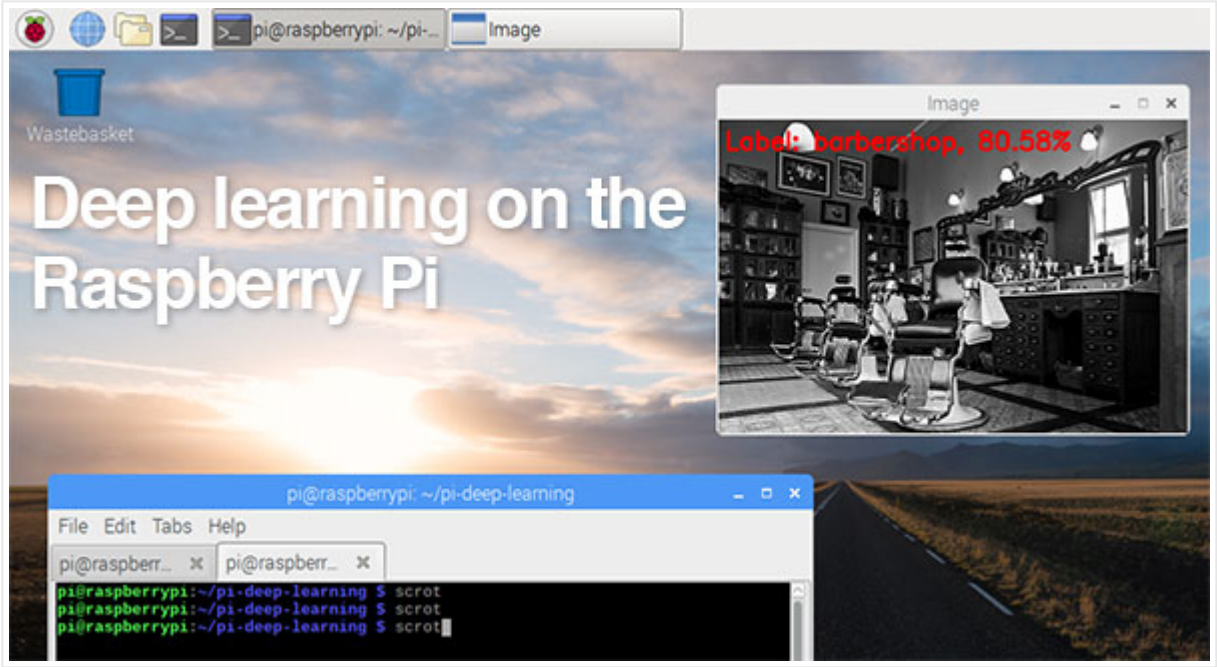
Deep learning on the Raspberry Pi with OpenCV

by **Adrian Rosebrock** on October 2, 2017 in **Deep Learning**, **Machine Learning**



I've received a number of emails from PyImageSearch readers who are interested in performing deep learning in their Raspberry Pi. Most of the questions go something like this:

> *Hey Adrian, thanks for all the tutorials on deep learning. You've really made deep learning accessible and easy to understand. I have a question: Can I do deep learning on the Raspberry Pi? What are the steps?*

And almost always, I have the same response:

> *The question really depends on what you mean by "do". You should never be training a neural network on the Raspberry Pi — it's far too underpowered. You're much better off training the network on your laptop, desktop, or even GPU (if you have one available).*
>
> *That said, you can deploy efficient, shallow neural networks to the Raspberry Pi and use them to classify input images.*

Again, I cannot stress this point enough:

You ***should not*** be training neural networks on the Raspberry Pi (unless you're using the Pi to do the *"Hello, World"* equivalent of neural networks — but again, I would still argue that your laptop/desktop is a better fit).

With the Raspberry Pi there just isn't enough RAM.

The processor is too slow.

And in general it's not the right hardware for heavy computational processes.

Instead, you should first ***train*** your network on your laptop, desktop, or deep learning environment.

Once the network is trained, you can then ***deploy*** the neural network to your Raspberry Pi.

In the remainder of this blog post I'll demonstrate how we can use the Raspberry Pi and pre- trained deep learning neural networks to classify input images.

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

Deep learning on the Raspberry Pi with OpenCV

When using the Raspberry Pi for deep learning we have two major pitfalls working against us:

1. Restricted memory (only 1GB on the Raspberry Pi 3).
2. Limited processor speed.

This makes it near impossible to use larger, deeper neural networks.

Instead, we need to use more computationally efficient networks with a smaller memory/processing footprint such as MobileNet and SqueezeNet. These networks are more appropriate for the Raspberry Pi; however, you need to set your expectations accordingly — you *should not* expect blazing fast speed.

In this tutorial we'll specifically be using SqueezeNet.

What is SqueezeNet?

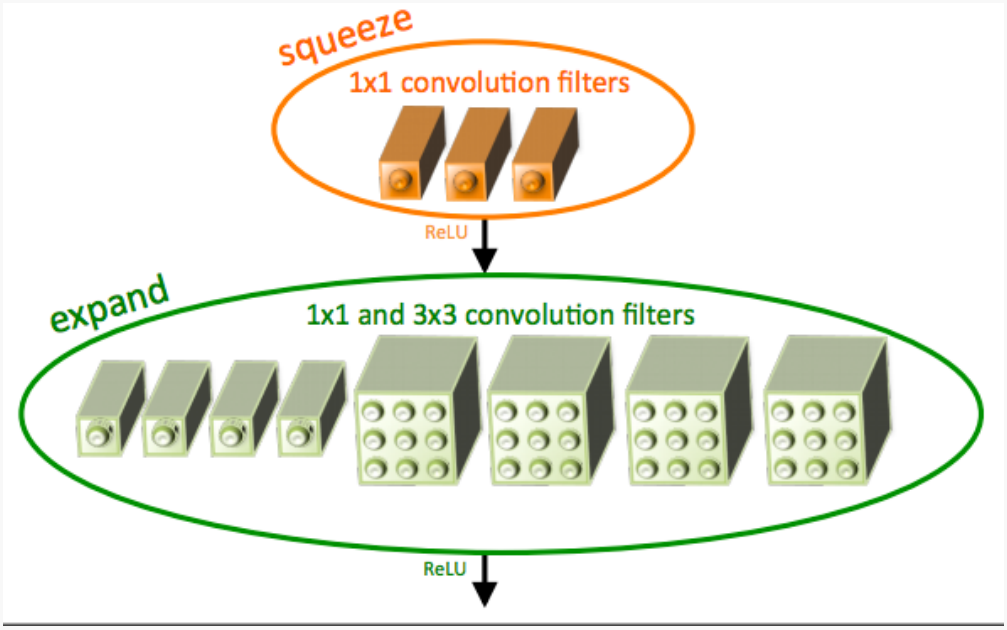


Figure 1: The "fire" module in SqueezeNet, consisting of a "squeeze" and an "expand" (Iandola et al., 2016).

SqueezeNet was first introduced by Iandola et al. in their 2016 paper, *SqueezeNet: AlexNet-level accuracy with 50x few parameters and <0.5MB model size*.

The title alone of this paper should pique your interest.

State-of-the-art architectures such as ResNet have model sizes that are >100MB. VGGNet is over 550MB. AlexNet sits in the middle of this size range with a model size of ~250MB.

In fact, one of the smaller Convolutional Neural Networks used for image classification is GoogLeNet at ~25-50MB (depending on which version of the architecture is implemented).

The real question is: *Can we go smaller?*

As the work of Iandola et al. demonstrates, the answer is: Yes, we can decrease model size by applying a novel usage of *1×1* and *3×3*convolutions, along with no fully-connected layers. The end result is a model weighing in at 4.9MB, which can be further reduced to < 0.5MB by model processing (also called "weight pruning" and "sparsifying a model").

In the remainder of this tutorial I'll be demonstrating how SqueezeNet can classify images in approximately half the time of GoogLeNet, making it a reasonable choice when applying deep learning on your Raspberry Pi.

Interested in learning more about SqueezeNet?

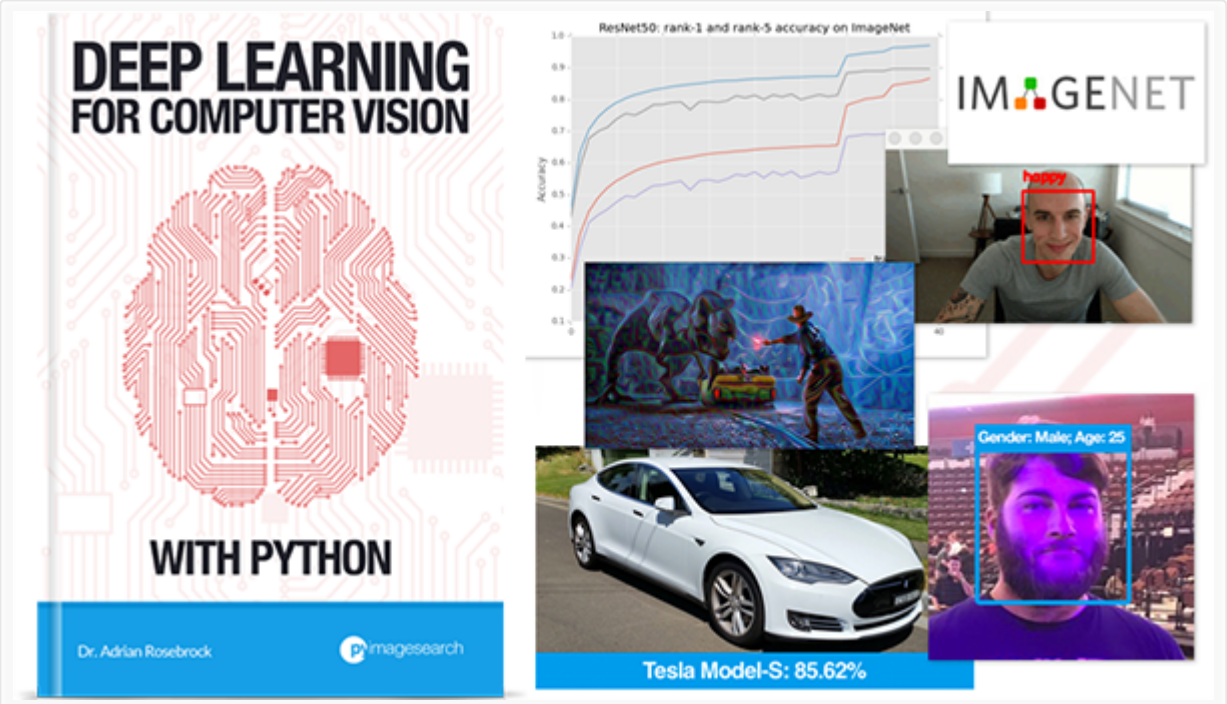


Figure 2: *Deep Learning for Computer Vision with Python* book

If you're interested in learning more about SqueezeNet, I would encourage you to take a look at my new book, *Deep Learning for Computer Vision with Python*.

Inside the *ImageNet Bundle*, I:

1. Explain the inner workings of the SqueezeNet architecture.
2. Demonstrate how to implement SqueezeNet by hand.
3. Train SqueezeNet from scratch on the challenging ImageNet dataset and replicate the original results by Iandola et al.

Go ahead and take a look — I think you'll agree with me when I say that this is the most complete deep learning + computer vision education you can find online.

Running a deep neural network on the Raspberry Pi

The source code from this blog post is heavily based on my previous post, *Deep learning with OpenCV*.

I'll still review the code in its entirety here; however, I would like to refer you over to the previous post for a complete and exhaustive review.

To get started, create a new file named `pi_deep_learning.py` , and insert the following source code:

Deep learning on the Raspberry Pi with Open(Python

```
# import the necessary packages
import numpy as np
import argparse
import time
import cv2
```

Lines 2-5 simply import our required packages.

From there, we need to parse our command line arguments:

```
Deep learning on the Raspberry Pi with Open(                    Python
 7  # construct the argument parse and parse the arguments
 8  ap = argparse.ArgumentParser()
 9  ap.add_argument("-i", "--image", required=True,
10      help="path to input image")
11  ap.add_argument("-p", "--prototxt", required=True,
12      help="path to Caffe 'deploy' prototxt file")
13  ap.add_argument("-m", "--model", required=True,
14      help="path to Caffe pre-trained model")
15  ap.add_argument("-l", "--labels", required=True,
16      help="path to ImageNet labels (i.e., syn-sets)")
17  args = vars(ap.parse_args())
```

As is shown on **Lines 9-16** we have four *required* command line arguments:

- `--image` : The path to the input image.
- `--prototxt` : The path to a Caffe prototxt file which is essentially a plaintext configuration file following a JSON-like structure. I cover the anatomy of Caffe projects in my PyImageSearch Gurus course.
- `--model` : The path to a pre-trained Caffe model. As stated above, you'll want to train your model on hardware which packs much more punch than the Raspberry Pi — we can, however, leverage a small, pre-existing model on the Pi.
- `--labels` : The path to class labels, in this case ImageNet "syn-sets" labels.

Next, we'll load the class labels and input image from disk:

```
Deep learning on the Raspberry Pi with Open(                    Python
19  # load the class labels from disk
20  rows = open(args["labels"]).read().strip().split("\n")
21  classes = [r[r.find(" ") + 1:].split(",")[0] for r in rows]
22
23  # load the input image from disk
24  image = cv2.imread(args["image"])
```

Go ahead and open `synset_words.txt` found in the ***"Downloads"***section of this post. You'll see on each line/row there is an ID and class labels associated with it (separated by commas).

Lines 20 and 21 simply read in the labels file line-by-line (`rows`) and extract the first relevant class label. The result is a `classes` list containing our class labels.

Then, we utilize OpenCV to load the image on **Line 24**.

Now we'll make use of OpenCV 3.3's Deep Neural Network (DNN) module to convert the `image` to a `blob` as well as to load the model from disk:

```
Deep learning on the Raspberry Pi with Open(                    Python
26  # our CNN requires fixed spatial dimensions for our input image(
27  # so we need to ensure it is resized to 227x227 pixels while
28  # performing mean subtraction (104, 117, 123) to normalize the i
29  # after executing this command our "blob" now has the shape:
30  # (1, 3, 227, 227)
31  blob = cv2.dnn.blobFromImage(image, 1, (227, 227), (104, 117, 12
32
33  # load our serialized model from disk
```

```
34 print("[INFO] loading model...")
35 net = cv2.dnn.readNetFromCaffe(args["prototxt"], args["model"])
```

Be sure to make note of the comment preceding our call to `cv2.dnn.blobFromImage` on **Line 31** above.

Common choices for width and height image dimensions inputted to Convolutional Neural Networks include 32 × 32, 64 × 64, 224 × 224, 227 × 227, 256 × 256, and 299 × 299. In our case we are pre-processing (normalizing) the image to dimensions of 227 x 227 (which are the image dimensions SqueezeNet was trained on) and performing a scaling technique known as mean subtraction. I discuss the importance of these steps in my book.

***Note:** You'll want to use 224 x 224 for the blob size when using **SqueezeNet** and 227 x 227 for **GoogLeNet** to be consistent with the prototxt definitions.*

We then load the network from disk on **Line 35** by utilizing our `prototxt` and `model` file path references.

In case you missed it above, it is worth noting here that we are loading a *pre-trained* model. The training step has already been performed on a more powerful machine and is outside the scope of this blog post (but covered in detail in both PyImageSearch Gurus and *Deep Learning for Computer Vision with Python*).

Now we're ready to pass the image through the network and look at the predictions:

```
Deep learning on the Raspberry Pi with Open(                    Python
37 # set the blob as input to the network and perform a forward-pas
38 # obtain our output classification
39 net.setInput(blob)
40 start = time.time()
41 preds = net.forward()
42 end = time.time()
43 print("[INFO] classification took {:.5} seconds".format(end - st
44
45 # sort the indexes of the probabilities in descending order (hig
46 # probabilitiy first) and grab the top-5 predictions
47 preds = preds.reshape((1, len(classes)))
48 idxs = np.argsort(preds[0])[::-1][:5]
```

To classify the query `blob`, we pass it forward through the network (**Lines 39-42**) and print out the amount of time it took to classify the input image (**Line 43**).

We can then sort the probabilities from highest to lowest (**Line 47**) while grabbing the top five `predictions` (**Line 48**).

The remaining lines (1) draw the highest predicted class label and corresponding probability on the image, (2) print the top five results and probabilities to the terminal, and (3) display the image to the screen:

```
Deep learning on the Raspberry Pi with Open(                    Python
50 # loop over the top-5 predictions and display them
51 for (i, idx) in enumerate(idxs):
52     # draw the top prediction on the input image
53     if i == 0:
54         text = "Label: {}, {:.2f}%".format(classes[idx],
55             preds[0][idx] * 100)
56         cv2.putText(image, text, (5, 25), cv2.FONT_HERSHEY_SIMPL
57             0.7, (0, 0, 255), 2)
```

```
58
59      # display the predicted label + associated probability to th
60      # console
61      print("[INFO] {}. label: {}, probability: {:.5}".format(i +
62          classes[idx], preds[0][idx]))
63
64  # display the output image
65  cv2.imshow("Image", image)
66  cv2.waitKey(0)
```

We draw the top prediction and probability on the top of the image (**Lines 53-57**) and display the top-5 predictions + probabilities on the terminal (**Lines 61 and 62**).

Finally, we display the output image on the screen (**Lines 65 and 66**). If you are using SSH to connect with your Raspberry Pi this will only work if you supply the `-X` flag for X11 forwarding when SSH'ing into your Pi.

To see the results of applying deep learning on the Raspberry Pi using OpenCV and Python, proceed to the next section.

Raspberry Pi and deep learning results

We'll be benchmarking our Raspberry Pi for deep learning against two pre-trained deep neural networks:

- GoogLeNet
- SqueezeNet

As we'll see, SqueezeNet is much smaller than GoogLeNet (5MB vs. 25MB, respectively) and will enable us to classify images substantially faster on the Raspberry Pi.

To run pre-trained Convolutional Neural Networks on the Raspberry Pi use the ***"Downloads"*** section of this blog post to download the source code + pre-trained neural networks + example images.

From there, let's first benchmark GoogLeNet against this input image:

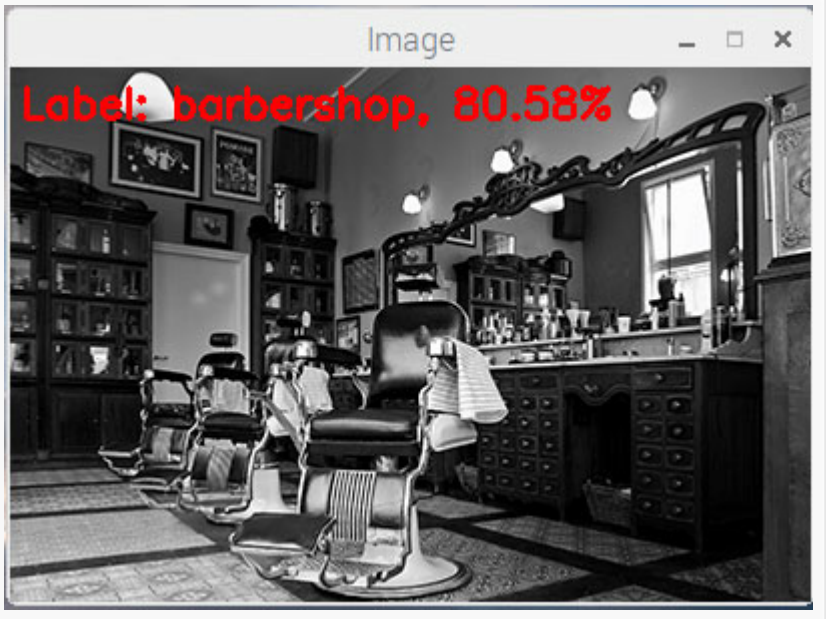


Figure 3: A *"barbershop"* is correctly classified by both GoogLeNet and Squeezenet using deep learning and OpenCV.

As we can see from the output, GoogLeNet correctly classified the image as *"barbershop"* in **1.7 seconds**:

```
Deep learning on the Raspberry Pi with OpenCV                    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 1.7304 seconds
[INFO] 1. label: barbershop, probability: 0.70508
[INFO] 2. label: barber chair, probability: 0.29491
[INFO] 3. label: restaurant, probability: 2.9732e-06
[INFO] 4. label: desk, probability: 2.06e-06
[INFO] 5. label: rocking chair, probability: 1.7565e-06
```

Let's give SqueezeNet a try:

```
Deep learning on the Raspberry Pi with OpenCV                    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 0.92073 seconds
[INFO] 1. label: barbershop, probability: 0.80578
[INFO] 2. label: barber chair, probability: 0.15124
[INFO] 3. label: half track, probability: 0.0052873
[INFO] 4. label: restaurant, probability: 0.0040124
[INFO] 5. label: desktop computer, probability: 0.0033352
```

SqueezeNet also correctly classified the image as *"barbershop"*…

…but in only 0.9 seconds!

As we can see, SqueezeNet is significantly faster than GoogLeNet — which is extremely important since we are applying deep learning to the resource constrained Raspberry Pi.

Let's try another example with SqueezeNet:

```
Deep learning on the Raspberry Pi with OpenCV                    Shell
$ python pi_deep_learning.py --prototxt models/squeezenet_v1.0.p
    --model models/squeezenet_v1.0.caffemodel --labels synset_wo
    --image images/cobra.png
[INFO] loading model...
[INFO] classification took 0.91687 seconds
[INFO] 1. label: Indian cobra, probability: 0.47972
[INFO] 2. label: leatherback turtle, probability: 0.16858
[INFO] 3. label: water snake, probability: 0.10558
[INFO] 4. label: common iguana, probability: 0.059227
[INFO] 5. label: sea snake, probability: 0.046393
```

Figure 4: SqueezeNet correctly classifies an image of a cobra using deep learning and OpenCV on the Raspberry Pi.

However, while SqueezeNet is significantly faster, it's less accurate than GoogLeNet:

Deep learning on the Raspberry Pi with OpenCV — Shell

```
$ python pi_deep_learning.py --prototxt models/squeezenet_v1.0.p
    --model models/squeezenet_v1.0.caffemodel --labels synset_wo
    --image images/jellyfish.png
[INFO] loading model...
[INFO] classification took 0.92117 seconds
[INFO] 1. label: bubble, probability: 0.59491
[INFO] 2. label: jellyfish, probability: 0.23758
[INFO] 3. label: Petri dish, probability: 0.13345
[INFO] 4. label: lemon, probability: 0.012629
[INFO] 5. label: dough, probability: 0.0025394
```

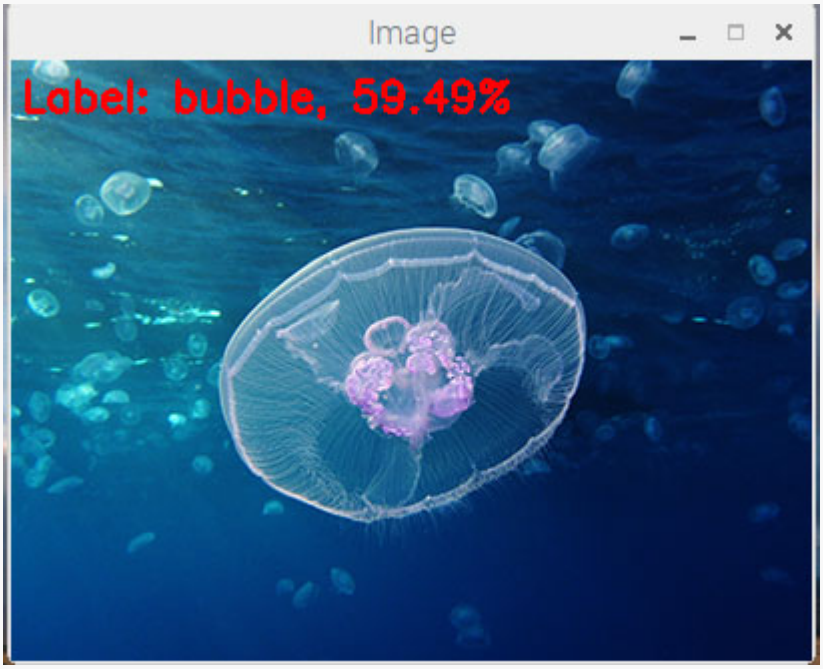


Figure 5: A jellyfish is incorrectly classified by SqueezNet as a bubble.

Here we see the top prediction by SqueezeNet is *"bubble"*. While the image may appear to have bubble-like characteristics, the image is actually of a *"jellyfish"* (which is the #2 prediction from SqueezeNet).

GoogLeNet on the other hand correctly reports *"jellyfish"* as the #1 prediction (with the sacrifice of processing time):

```
Deep learning on the Raspberry Pi with OpenC                    Shell
$ python pi_deep_learning.py --prototxt models/bvlc_googlenet.pr
    --model models/bvlc_googlenet.caffemodel --labels synset_wor
    --image images/jellyfish.png
[INFO] loading model...
[INFO] classification took 1.7824 seconds
[INFO] 1. label: jellyfish, probability: 0.53186
[INFO] 2. label: bubble, probability: 0.33562
[INFO] 3. label: tray, probability: 0.050089
[INFO] 4. label: shower cap, probability: 0.022811
[INFO] 5. label: Petri dish, probability: 0.013176
```

Summary

Today, we learned how to apply deep learning on the Raspberry Pi using Python and OpenCV.

In general, you should:

1. Never use your Raspberry Pi to *train* a neural network.
2. Only use your Raspberry Pi to *deploy* a pre-trained deep learning network.

The Raspberry Pi does not have enough memory or CPU power to train these types of deep, complex neural networks from scratch.

In fact, the Raspberry Pi *barely* has enough processing power to run them — as we'll find out in next week's blog post you'll struggle to get a reasonable frames per second for video processing applications.

If you're interested in embedded deep learning on low cost hardware, I'd consider looking at optimized devices such as NVIDIA's Jetson TX1 and TX2. These boards are designed to execute neural networks on the GPU and provide real-time (or as close to real-time as possible) classification speed.

In next week's blog post, I'll be discussing how to optimize OpenCV on the Raspberry Pi to obtain performance gains by ***upwards of 100%*** for object detection using deep learning.

To be notified when this blog post is published, *just enter your email address in the form below!*

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!

classification, **cnn**, **convolutional neural network**, **deep learning**, **machine learning**, **opencv**

27 Responses to *Deep learning on the Raspberry Pi with OpenCV*

Andrey October 2, 2017 at 10:55 am #

REPLY

Nice detailed post. Thank you.

Adrian Rosebrock October 3, 2017 at 11:00 am #

REPLY

Thanks Andrey!

Flávio Rodrigues October 2, 2017 at 11:03 am #

REPLY

Hi, Dr. Adrian! Should not the size of the input image be 227×227 as reported in the prototxt file? And just to be perfect as always, there is a typo on line "Train SqueezeNet from scratch on the challenging ImageNetd ataset and replicate the original results by Iandola et al." 😉 Thanks a lot for your posts!

Adrian Rosebrock October 3, 2017 at 11:02 am #

REPLY

Thank you for reporting the typo, Flávio! I have fixed it now.

As for SqueezeNet, yes, it should be 227×227 as that is what the prototxt reports. I have updated that typo as well.

Ashutosh Dubey October 2, 2017 at 12:39 pm #

REPLY

does your book “deep learning for computer vision” is useful to work with raspberry pi?

Adrian Rosebrock October 3, 2017 at 11:00 am #

REPLY

You can go through the vast majority of the Starter Bundle of Deep Learning for Computer Vision with Python on the Raspberry Pi, but not the Practitioner Bundle or ImageNet Bundle.

As I mentioned in the blog post you should really be using your laptop or desktop if you intend to train neural networks with scratch.

Cristian Benglenok October 2, 2017 at 12:49 pm #

REPLY

if I set a cluster with 2 or 4 raspbery pi 3, will increase the processing speed considerably? if we use it to deploy a pre-trained deep learning network.

Adrian Rosebrock October 3, 2017 at 10:59 am #

REPLY

Not really, because at that point you’ll be dealing with network I/O latency at inference time. You’ll need a faster system if you need quicker classifications.

fariborz October 2, 2017 at 1:16 pm #

REPLY

hi Adrian
tanx alot
nice post
i need deep learning with raspberry pi

please make a tutorial for Drowsiness detection with raspberry pi.
i do drowsiness project with raspberry but it is lagy and very slow

REPLY

Adrian Rosebrock October 3, 2017 at 10:57 am #

I will be doing a Raspberry Pi + drowsiness detector later this month (October 2017).

REPLY

fariborz October 8, 2017 at 4:40 pm #

wow nice tanx a lot

REPLY

Kiran October 2, 2017 at 2:26 pm #

That's so awesome! I can't believe we have made it this far with all the CV blogs! The Pi community is so big and getting bigger and bigger with DNNs. I wanted to know if there's a separate webpage on your website where people who follow your blog could benchmark and share all their Pi's performance ratings for CNNs! Looking forward to your upcoming post on running it on live stream! I pity those Pi's taking so much stress! ha ha 😛

REPLY

Adrian Rosebrock October 3, 2017 at 10:58 am #

Hi Kiran — I don't have any a page dedicated where readers can share their benchmarks using deep learning and the Raspberry Pi, but I'll definitely consider this for the future.

REPLY

aditya October 2, 2017 at 2:44 pm #

Hello Sir Please write a blog on how to make a model for object recognization in image steps to train the model and what algorithm using deep learning please please make such tut as you have made a tutorial for object detection but it has pre-trained model but i want to learn how to train manually the images soo make such blog tut…
As you have told before that this is available in your book but really i don't have enough money to buy your book but i always read your blog and gain knoweldge. I use collage resources to implement your code and i am really intrested in learning so i work hard but i don't have

enough money to get the paid course but your blog has helped me a lot soo thank you very much…

Aleksandr Rybnikov October 2, 2017 at 4:53 pm #

REPLY

Thank you, Adrian! Excellent job! I just want to add that model's size isn't an exhausting way to estimate inference time. For example, ENet architectire has ~10MB as all weights, but takes a lot of computations and time to apply these weights. OpenCV's dnn has functions to get FLOPs for model without inference.
Also recently I added new face detector to the dnn. It's ResNet-based SSD and I trained it on some open dataset full of faces. It works faster than real-time on i7-6700k and according to tests it's better than casscades-based one. Maybe this infirmation will be interesting for you

Adrian Rosebrock October 3, 2017 at 10:57 am #

REPLY

Hi Aleksandr — you're absolutely right, examining model size is not an exact way to measure inference time. I was instead referring to the limited RAM available on the Raspberry Pi and that users need to be careful not to exhaust this memory.

The new ResNet-based SSD face detector sounds very interesting as well! I'll be sure to take a look at this.

Abrie October 3, 2017 at 7:18 am #

REPLY

Hi Adrian. Thanks a lot for these tutorials.

Will next weeks blog post cover how to detect objects from a live video stream on the Raspberry Pi and Picam module?

Kind regards

Abrie

Adrian Rosebrock October 3, 2017 at 10:55 am #

REPLY

Hi Abrie — next week's blog post will be on optimizing OpenCV and milking every last bit of performance out of it. The following week will then cover deep learning-based object detection on the Raspberry Pi.