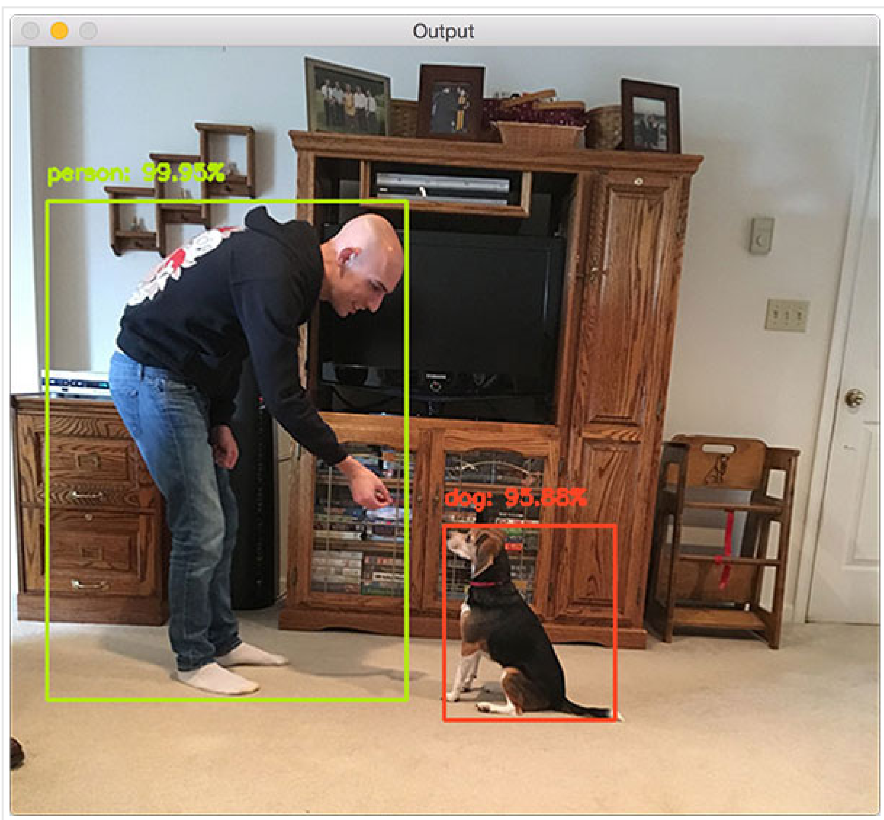
Object detection with deep learning and OpenCV

by Adrian Rosebrock on September 11, 2017 in Deep Learning, OpenCV 3, Tutorials



A couple weeks ago we learned how to classify images using deep learning and OpenCV 3.3's deep neural network (`dnn`) module.

While this original blog post demonstrated how we can ***categorize*** an image into one of ImageNet's 1,000 separate class labels it *could not* tell us ***where*** an object resides in image.

In order to obtain the bounding box *(x, y)*-coordinates for an object in a image we need to instead apply **object detection**.

Object detection can not only tell us ***what*** is in an image but also ***where*** the object is as well.

In the remainder of today's blog post we'll discuss how to apply object detection using deep learning and OpenCV.

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

Object detection with deep learning and OpenCV

In the first part of today's post on object detection using deep learning we'll discuss *Single Shot Detectors* and *MobileNets*.

When combined together these methods can be used for super fast, real-time object detection on resource constrained devices (including the Raspberry Pi, smartphones, etc.)

From there we'll discover how to use OpenCV's `dnn` module to load a pre-trained object detection network.

This will enable us to pass input images through the network and obtain the output bounding box *(x, y)*-coordinates of each object in the image.

Finally we'll look at the results of applying the MobileNet Single Shot Detector to example input images.

In a future blog post we'll extend our script to work with real-time video streams as well.

Single Shot Detectors for object detection

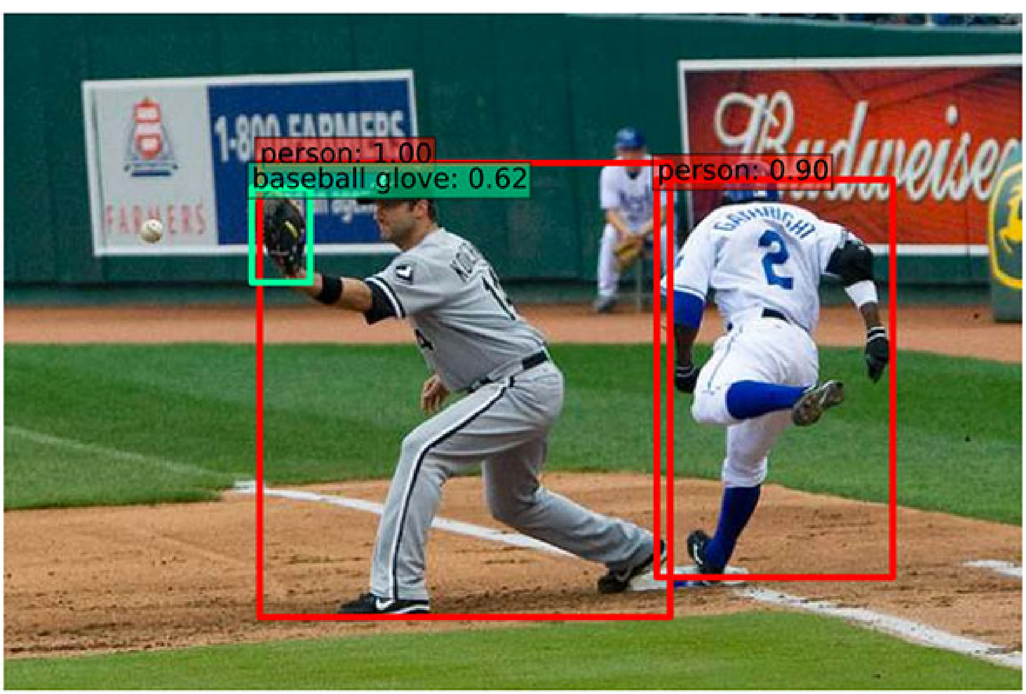


Figure 1: Examples of object detection using Single Shot Detectors (SSD) from Liu et al.

When it comes to deep learning-based object detection there are three primary object detection methods that you'll likely encounter:

- Faster R-CNNs (Girshick et al., 2015)
- You Only Look Once (YOLO) (Redmon and Farhadi, 2015)
- Single Shot Detectors (SSDs) (Liu et al., 2015)

Faster R-CNNs are likely the most "heard of" method for object detection using deep learning; however, the technique can be difficult to understand (especially for beginners in deep learning), hard to implement, and challenging to train.

Furthermore, even with the "faster" implementation R-CNNs (where the "R" stands for "Region Proposal") the algorithm can be quite slow, on the order of 7 FPS.

If we are looking for pure speed then we tend to use YOLO as this algorithm is much faster, capable of processing 40-90 FPS on a Titan X GPU. The super fast variant of YOLO can even get up to 155 FPS.

The problem with YOLO is that it leaves much accuracy to be desired.

SSDs, originally developed by Google, are a balance between the two. The algorithm is more straightforward (and I would argue better explained in the original seminal paper) than Faster R-CNNs.

We can also enjoy a much faster FPS throughput than Girshick et al. at 22-46 FPS depending on which variant of the network we use. SSDs also tend to be more accurate than YOLO. To learn more about SSDs, please refer to Liu et al.

MobileNets: Efficient (deep) neural networks

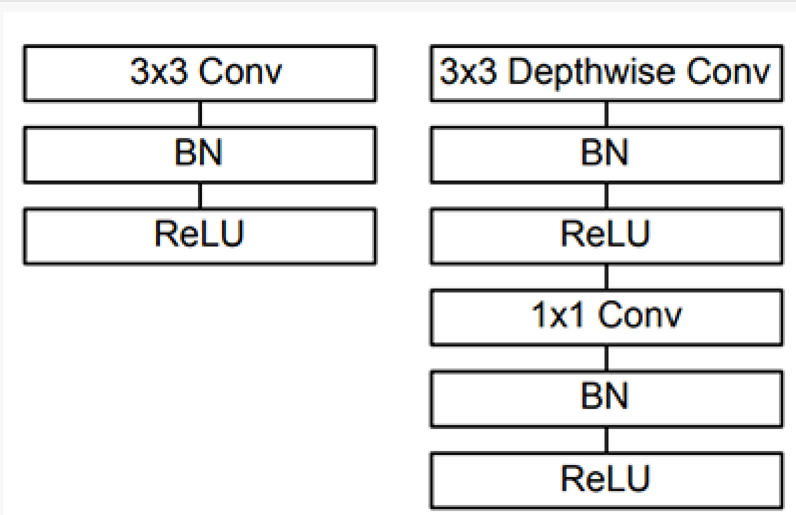


Figure 2: *(Left)* Standard convolutional layer with batch normalization and ReLU. *(Right)* Depthwise separable convolution with depthwise and pointwise layers followed by batch normalization and ReLU (figure and caption from Liu et al.).

When building object detection networks we normally use an existing network architecture, such as VGG or ResNet, and then use it inside the object detection pipeline. The problem is that these network architectures can be very large in the order of 200-500MB.

Network architectures such as these are unsuitable for resource constrained devices due to their sheer size and resulting number of computations.

Instead, we can use MobileNets (Howard et al., 2017), another paper by Google researchers. We call these networks “MobileNets” because they are designed for resource constrained devices such as your smartphone. MobileNets differ from traditional CNNs through the usage of *depthwise separable convolution* (**Figure 2** above).

The general idea behind depthwise separable convolution is to split convolution into two stages:

1. A *3×3* depthwise convolution.
2. Followed by a *1×1* pointwise convolution.

This allows us to actually reduce the number of parameters in our network.

The problem is that we sacrifice accuracy — MobileNets are normally not as accurate as their larger big brothers…

…but they are much more resource efficient.

For more details on MobileNets please see Howard et al.

Combining MobileNets and Single Shot Detectors for fast, efficient deep-learning based object detection

If we combine both the MobileNet architecture and the Single Shot Detector (SSD) framework, we arrive at a fast, efficient deep learning-based method to object detection.

The model we’ll be using in this blog post is a Caffe version of the original TensorFlow implementation by Howard et al. and was trained by chuanqi305 (see GitHub).

The MobileNet SSD was first trained on the COCO dataset (Common Objects in Context) and was then fine-tuned on PASCAL VOC reaching 72.7% mAP (mean average precision).

We can therefore detect 20 objects in images (+1 for the background class), including *airplanes, bicycles, birds, boats, bottles, buses, cars, cats, chairs, cows, dining tables, dogs, horses, motorbikes, people, potted plants, sheep, sofas, trains,* and *tv monitors*.

Deep learning-based object detection with OpenCV

In this section we will use the MobileNet SSD + deep neural network (`dnn`) module in OpenCV to build our object detector.

I would suggest using the ***"Downloads"*** code at the bottom of this blog post to download the source code + trained network + example images so you can test them on your machine.

Let's go ahead and get started building our deep learning object detector using OpenCV.

Open up a new file, name it `deep_learning_object_detection.py` , and insert the following code:

Object detection with deep learning and OpenCV — Python

```python
1  # import the necessary packages
2  import numpy as np
3  import argparse
4  import cv2
5  
6  # construct the argument parse and parse the arguments
7  ap = argparse.ArgumentParser()
8  ap.add_argument("-i", "--image", required=True,
9      help="path to input image")
10 ap.add_argument("-p", "--prototxt", required=True,
11     help="path to Caffe 'deploy' prototxt file")
12 ap.add_argument("-m", "--model", required=True,
13     help="path to Caffe pre-trained model")
14 ap.add_argument("-c", "--confidence", type=float, default=0.2,
15     help="minimum probability to filter weak detections")
16 args = vars(ap.parse_args())
```

On **Lines 2-4** we import packages required for this script — the `dnn` module is included in `cv2` , again, making hte assumption that you're using *OpenCV 3.3*.

Then, we parse our command line arguments (**Lines 7-16**):

- `--image` : The path to the input image.
- `--prototxt` : The path to the Caffe prototxt file.
- `--model` : The path to the pre-trained model.
- `--confidence` : The minimum probability threshold to filter weak detections. The default is 20%.

Again, example files for the first three arguments are included in the ***"Downloads"*** section of this blog post. I urge you to start there while also supplying some query images of your own.

Next, let's initialize class labels and bounding box colors:

Object detection with deep learning and OpenCV — Python

```python
18 # initialize the list of class labels MobileNet SSD was trained to
19 # detect, then generate a set of bounding box colors for each class
20 CLASSES = ["background", "aeroplane", "bicycle", "bird", "boat",
21     "bottle", "bus", "car", "cat", "chair", "cow", "diningtable",
22     "dog", "horse", "motorbike", "person", "pottedplant", "sheep",
23     "sofa", "train", "tvmonitor"]
24 COLORS = np.random.uniform(0, 255, size=(len(CLASSES), 3))
```

Lines 20-23 build a list called `CLASSES` containing our labels. This is followed by a list, `COLORS` which contains corresponding random colors for bounding boxes (**Line 24**).

Now we need to load our model:

```
26 # load our serialized model from disk
27 print("[INFO] loading model...")
28 net = cv2.dnn.readNetFromCaffe(args["prototxt"], args["model"])
```

The above lines are self-explanatory, we simply print a message and load our `model` (**Lines 27 and 28**).

Next, we will load our query image and prepare our `blob`, which we will feed-forward through the network:

Object detection with deep learning and OpenCV — Python

```
30 # load the input image and construct an input blob for the image
31 # by resizing to a fixed 300x300 pixels and then normalizing it
32 # (note: normalization is done via the authors of the MobileNet SSD
33 # implementation)
34 image = cv2.imread(args["image"])
35 (h, w) = image.shape[:2]
36 blob = cv2.dnn.blobFromImage(image, 0.007843, (300, 300), 127.5)
```

Taking note of the comment in this block, we load our `image` (**Line 34**), extract the height and width (**Line 35**), and calculate a *300 by 300* pixel `blob` from our image (**Line 36**).

Now we're ready to do the heavy lifting — we'll pass this blob through the neural network:

Object detection with deep learning and OpenCV — Python

```
38 # pass the blob through the network and obtain the detections and
39 # predictions
40 print("[INFO] computing object detections...")
41 net.setInput(blob)
42 detections = net.forward()
```

On **Lines 41 and 42** we set the input to the network and compute the forward pass for the input, storing the result as `detections`. Computing the forward pass and associated detections could take awhile depending on your model and input size, but for this example it will be relatively quick on most CPUs.

Let's loop through our `detections` and determine *what* and *where* the objects are in the image:

Object detection with deep learning and OpenCV — Python

```
# loop over the detections
for i in np.arange(0, detections.shape[2]):
    # extract the confidence (i.e., probability) associated with the
    # prediction
    confidence = detections[0, 0, i, 2]

    # filter out weak detections by ensuring the `confidence` is
    # greater than the minimum confidence
    if confidence > args["confidence"]:
        # extract the index of the class label from the `detections`,
        # then compute the (x, y)-coordinates of the bounding box for
        # the object
        idx = int(detections[0, 0, i, 1])
        box = detections[0, 0, i, 3:7] * np.array([w, h, w, h])
        (startX, startY, endX, endY) = box.astype("int")

        # display the prediction
        label = "{}: {:.2f}%".format(CLASSES[idx], confidence * 100)
        print("[INFO] {}".format(label))
        cv2.rectangle(image, (startX, startY), (endX, endY),
```

```
64                  COLORS[idx], 2)
65              y = startY - 15 if startY - 15 > 15 else startY + 15
66              cv2.putText(image, label, (startX, y),
67                  cv2.FONT_HERSHEY_SIMPLEX, 0.5, COLORS[idx], 2)
```

We start by looping over our detections, keeping in mind that multiple objects can be detected in a single image. We also apply a check to the confidence (i.e., probability) associated with each detection. If the confidence is high enough (i.e. above the threshold), then we'll display the prediction in the terminal as well as draw the prediction on the image with text and a colored bounding box. Let's break it down line-by-line:

Looping through our `detections` , first we extract the `confidence` value (**Line 48**).

If the `confidence` is above our minimum threshold (**Line 52**), we extract the class label index (**Line 56**) and compute the bounding box around the detected object (**Line 57**).

Then, we extract the *(x, y)*-coordinates of the box (**Line 58**) which we will will use shortly for drawing a rectangle and displaying text.

Next, we build a text `label` containing the `CLASS` name and the `confidence` (**Line 61**).

Using the label, we print it to the terminal (**Line 62**), followed by drawing a colored rectangle around the object using our previously extracted *(x, y)*-coordinates (**Lines 63 and 64**).

In general, we want the label to be displayed above the rectangle, but if there isn't room, we'll display it just below the top of the rectangle (**Line 65**).

Finally, we overlay the colored text onto the `image` using the *y*-value that we just calculated (**Lines 66 and 67**).

The only remaining step is to display the result:

Object detection with deep learning and OpenCV — Python

```python
69 # show the output image
70 cv2.imshow("Output", image)
71 cv2.waitKey(0)
```

We display the resulting output image to the screen until a key is pressed (**Lines 70 and 71**).

OpenCV and deep learning object detection results

To download the code + pre-trained network + example images, be sure to use the ***"Downloads"*** section at the bottom of this blog post.

From there, unzip the archive and execute the following command:

Object detection with deep learning and OpenCV — Shell

```shell
$ python deep_learning_object_detection.py \
    --prototxt MobileNetSSD_deploy.prototxt.txt \
    --model MobileNetSSD_deploy.caffemodel --image images/example_01.jpg
[INFO] loading model...
```

```
[INFO] computing object detections...
[INFO] loading model...
[INFO] computing object detections...
[INFO] car: 99.78%
[INFO] car: 99.25%
```

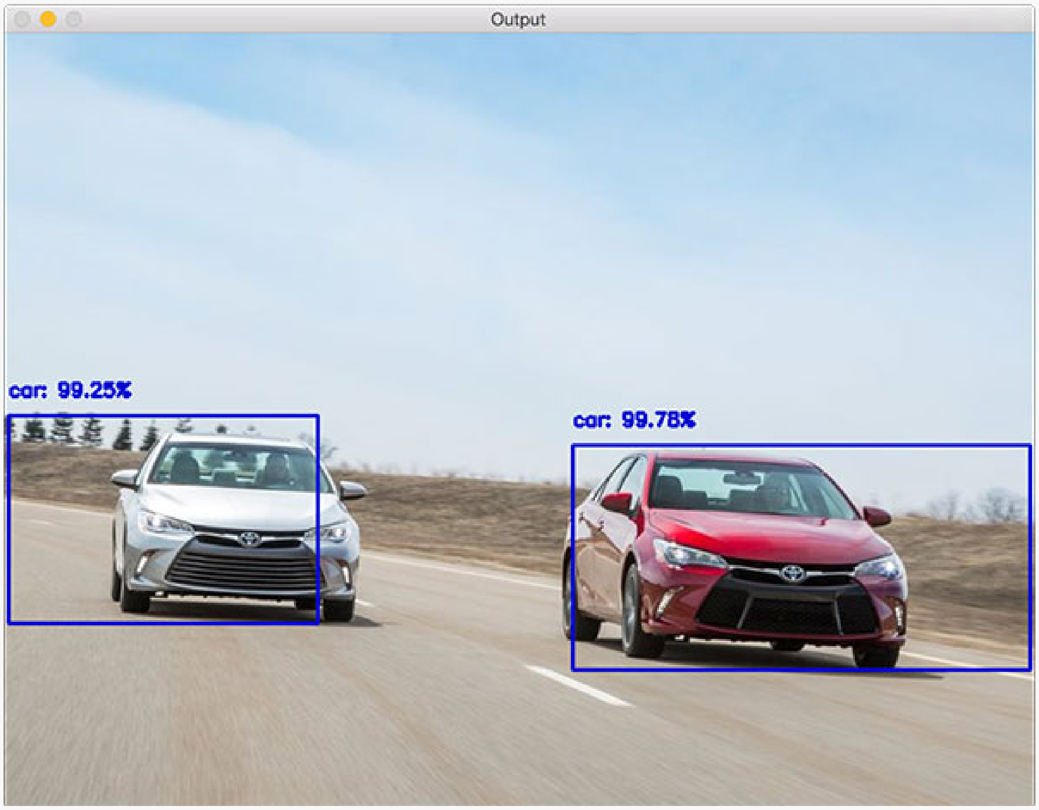


Figure 3: Two Toyotas on the highway recognized with near-100% confidence using OpenCV, deep learning, and object detection.

Our first result shows cars recognized and detected with near-100% confidence.

In this example we detect an airplane using deep learning-based object detection:

Object detection with deep learning and OpenCV — Shell

```
$ python deep_learning_object_detection.py \
    --prototxt MobileNetSSD_deploy.prototxt.txt \
    --model MobileNetSSD_deploy.caffemodel --image images/example_02.jpg
[INFO] loading model...
[INFO] computing object detections...
[INFO] loading model...
[INFO] computing object detections...
[INFO] aeroplane: 98.42%
```

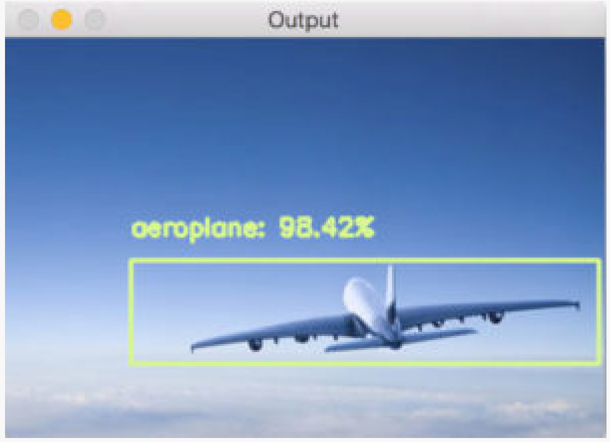


Figure 4: An airplane successfully detected with high confidence via Python, OpenCV, and deep learning.

The ability for deep learning to detect and localize obscured objects is demonstrated in the following image, where we see a horse (and it's rider) jumping a fence flanked by two potted plants:

Object detection with deep learning and OpenCV

```
$ python deep_learning_object_detection.py \
    --prototxt MobileNetSSD_deploy.prototxt.txt \
    --model MobileNetSSD_deploy.caffemodel --image images/example_03.jpg
[INFO] loading model...
[INFO] computing object detections...
[INFO] horse: 96.67%
[INFO] person: 92.58%
[INFO] pottedplant: 96.87%
[INFO] pottedplant: 34.42%
```

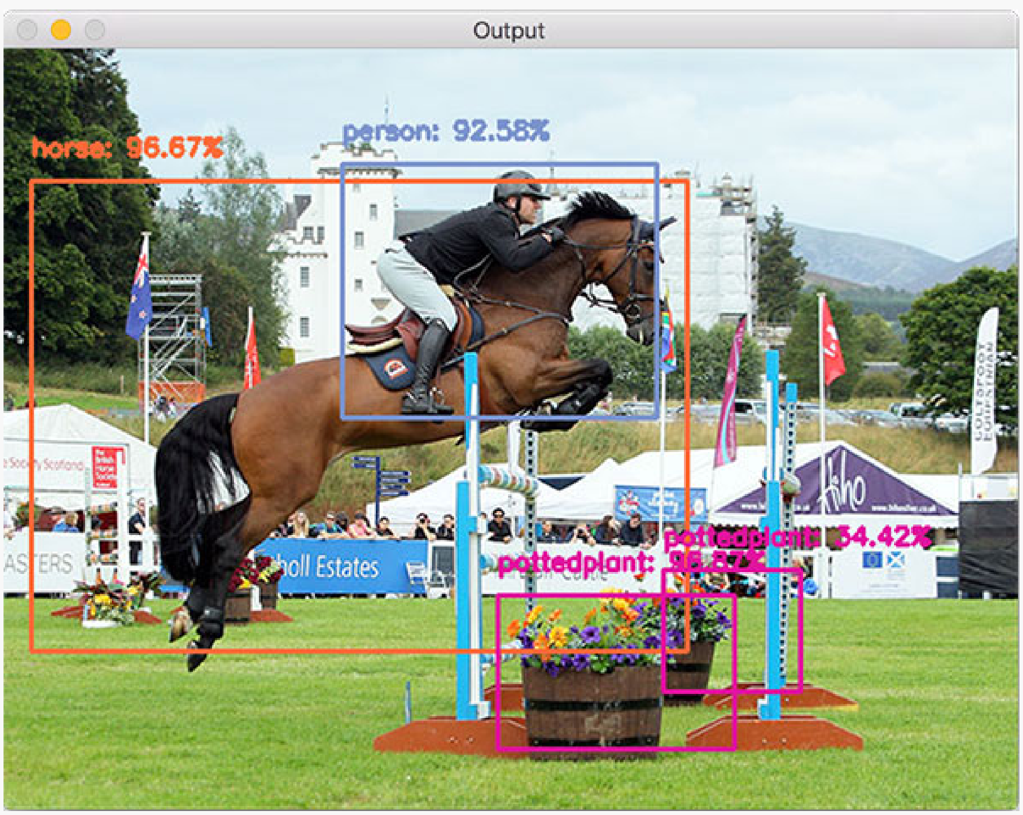


Figure 5: A person riding a horse and two potted plants are successfully identified despite a lot of objects in the image via deep learning-based object detection.

In this example we can see a beer bottle is detected with an impressive 100% confidence:

Object detection with deep learning and OpenCV — Python

```
$ python deep_learning_object_detection.py --prototxt MobileNetSSD_deploy.prototxt.txt \
    --model MobileNetSSD_deploy.caffemodel --image images/example_04.jpg
[INFO] loading model...
[INFO] computing object detections...
[INFO] bottle: 100.00%
```

Figure 6: Deep learning + OpenCV are able to correctly detect a beer bottle in an input image.

Followed by another horse image which also contains a dog, car, and person:

Object detection with deep learning and OpenCV — Shell

```
$ python deep_learning_object_detection.py \
    --prototxt MobileNetSSD_deploy.prototxt.txt \
    --model MobileNetSSD_deploy.caffemodel --image images/example_05.jpg
[INFO] loading model...
[INFO] computing object detections...
[INFO] car: 99.87%
[INFO] dog: 94.88%
[INFO] horse: 99.97%
[INFO] person: 99.88%
```

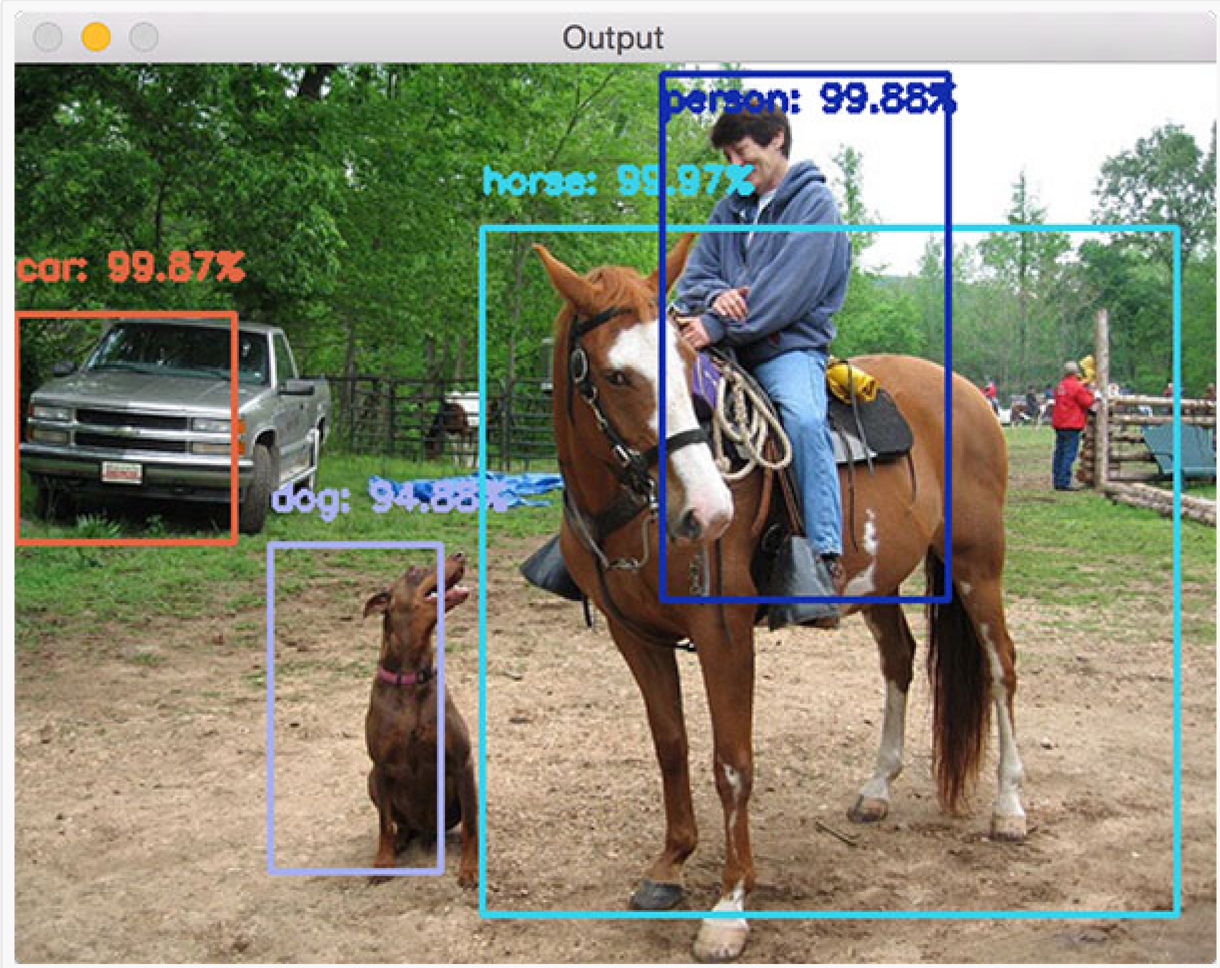


Figure 7: Several objects in this image including a car, dog, horse, and person are all recognized.

Finally, a picture of me and Jemma, the family beagle:

Object detection with deep learning and OpenCV

```
$ python deep_learning_object_detection.py \
    --prototxt MobileNetSSD_deploy.prototxt.txt \
    --model MobileNetSSD_deploy.caffemodel --image images/example_06.jpg
[INFO] loading model...
[INFO] computing object detections...
[INFO] dog: 95.88%
[INFO] person: 99.95%
```

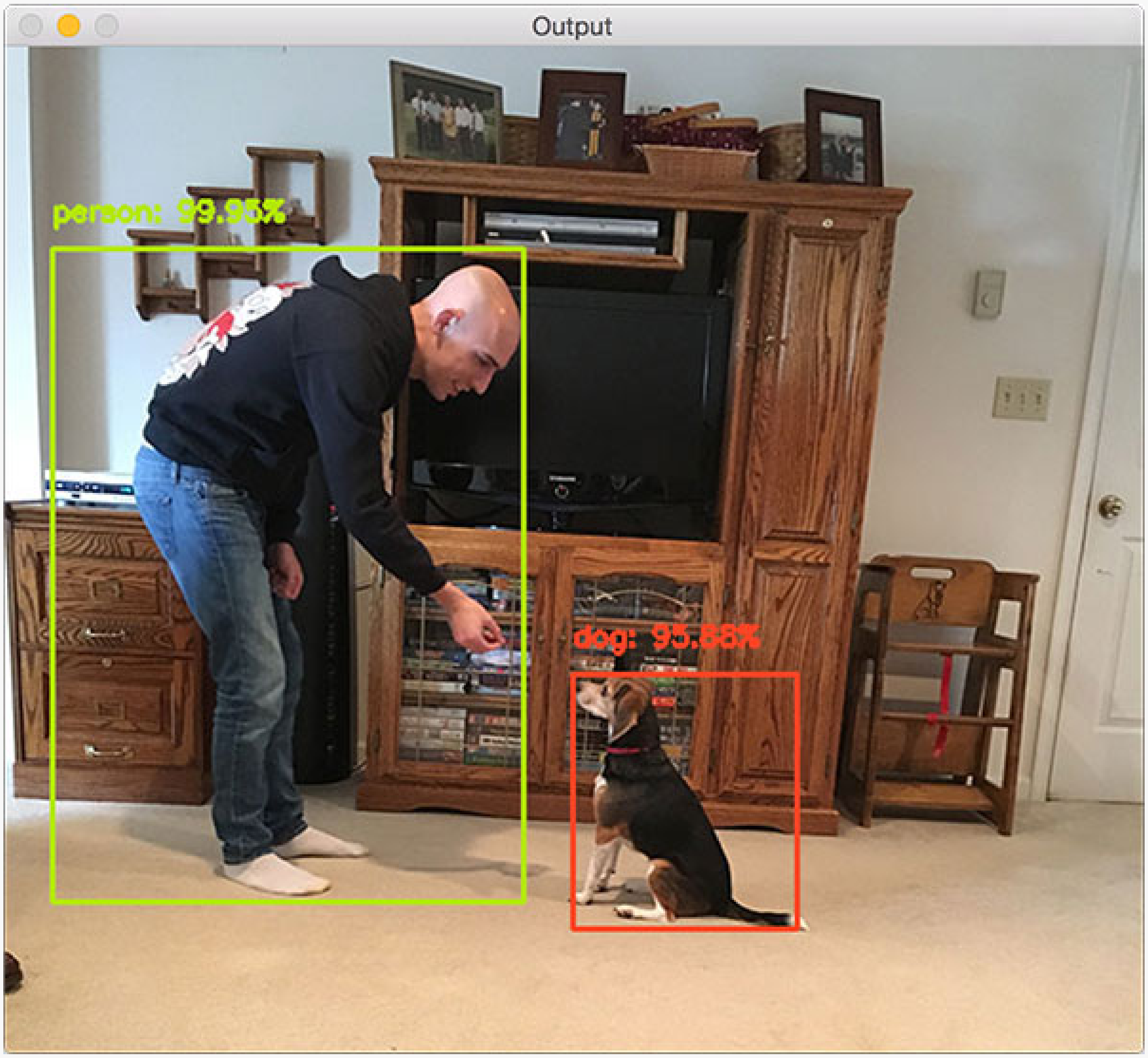


Figure 8: Me and the family beagle are corrected as a *"person"* and a *"dog"* via deep learning, object detection, and OpenCV. The TV monitor is not recognized.

Unfortunately the TV monitor isn't recognized in this image which is likely due to (1) me blocking it and (2) poor contrast around the TV. That being said, we have demonstrated excellent object detection results using OpenCV's `dnn` module.

Summary

In today's blog post we learned how to perform object detection using deep learning and OpenCV.

Specifically, we used both *MobileNets* + *Single Shot Detectors* along with OpenCV 3.3's brand new (totally overhauled) `dnn` module to detect objects in images.

As a computer vision and deep learning community we owe a lot to the contributions of Aleksandr Rybnikov, the main contributor to the `dnn` module for making deep learning so accessible from within the OpenCV library. You can find Aleksandr's original OpenCV example script here — I have modified it for the purposes of this blog post.

In a future blog post I'll be demonstrating how we can modify today's tutorial to work with real-time video streams, thus enabling us to perform deep learning-based object detection to videos. We'll be sure to leverage efficient frame I/O to increase the FPS throughout our pipeline as well.

To be notified when future blog posts (such as the real-time object detection tutorial) are published here on PyImageSearch, *simply enter your email address in the form below*.

Downloads:

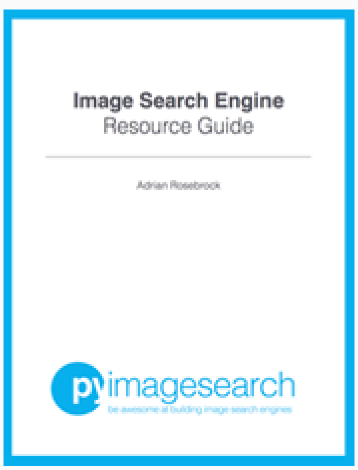
If you would like to download the code and images used in this post, please enter your email address in the form below. Not only will you get a .zip of the code, I'll also send you a **FREE 11-page Resource Guide** on Computer Vision and Image Search Engines, including **exclusive techniques** that I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

Email address:

Your email address

DOWNLOAD THE CODE!

Resource Guide (it's totally free).



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

Your email address

DOWNLOAD THE GUIDE!

cnn, **coco**, **convolutional neural network**, **deep learning**, **machine learning**, **opencv 3**, **pascal voc**, **real-time**, **video**, **video stream**

‹ Raspbian Stretch: Install OpenCV 3 + Python on your Raspberry Pi

43 Responses to *Object detection with deep learning and OpenCV*

tommy September 11, 2017 at 11:41 am #

REPLY

how do we train the dnn using opencv or do we have to use tensorflow and the likes?

plus where can we get some sample caffemodels?

tensorflow has some models in its own ckpt format.

Adrian Rosebrock September 11, 2017 at 2:31 pm #

REPLY

I would start by giving the first post in the series a read. You do not train the models with OpenCV's dnn module. They are instead trained using tools like Caffe, TensorFlow, or PyTorch. This particular example demonstrates how to load a pre-trained Caffe network.

The dnn module has been totally re-done in OpenCV 3.3. Many Caffe models will work with it out-of-the-box. I would suggest taking a look at the Caffe Model Zoo for more pre-trained networks.

Max September 11, 2017 at 11:46 am #

REPLY

Hi Adrian,
how long does it take to forward walk through the provided network?
Is it faster than tensorflow based networks of same architecture?
Is there a tutorial inside of your books that covers fast recognition and detection using CNN at best in realtime with networks like YOLO.

Adrian Rosebrock September 11, 2017 at 2:29 pm #

REPLY

1. As I'll be discussing in next week's tutorial you'll be able to get 6-8 frames per second using this method.

2. Once the model is trained you won't see massive speed increases as it's (1) just the forward pass and (2) OpenCV is loading the serialized weights from disk.

3. Yes, I will be covering object detection inside Deep Learning for Computer Vision with Python. You'll want to go with the ImageNet Bundle where I discuss SSD and Faster R-CNNs.

Vasanth September 11, 2017 at 1:00 pm #

REPLY

Hi Adrian , You always inspired me with your Tremendous Innovation and become my Role Model too….

Now Coming back to the Topic , I'm Getting this error :

Traceback (most recent call last):
File "deep_learning_object_detection.py", line 32, in
net = cv2.dnn.readNetFromCaffe(args["prototxt"], args["model"])
AttributeError: 'module' object has no attribute 'dnn'

Eventhough after installing Lasagne , it is giving me the error :
ImportError: Could not import Theano.

Please make sure you install a recent enough version of Theano. See
section 'Install from PyPI' in the installation docs for more details:
http://lasagne.readthedocs.org/en/latest/user/installation.html#install-from-pypi

Adrian Rosebrock September 11, 2017 at 2:26 pm #

REPLY

Hi Vasanth — you need to install **OpenCV 3.3** for this tutorial to work. Lasange and Theano **are not needed** and you can safely skip them.

andrew September 11, 2017 at 1:19 pm #

REPLY

Great post, It makes me even more excited for your deep learning book

Adrian Rosebrock September 11, 2017 at 2:24 pm #

REPLY

Thanks Andrew — I'll be sharing how to train your own custom object detector inside Deep Learning for Computer Vision with Python as well.

aditya September 11, 2017 at 1:32 pm #

REPLY

Can you please provide the dataset link and the train.py file
i want to manually train it and check it…
So please provide the dataset name or downloading link and the program to train the model…

Adrian Rosebrock September 11, 2017 at 2:25 pm #

REPLY

Hi Aditya — as I mentioned in the tutorial this object detector is pre-trained via the Caffe framework. I'll be discussing hwo to create your own custom object detectors inside Deep Learning for Computer Vision with Python.

Sydney September 11, 2017 at 3:56 pm #

REPLY

Nice tutorial. Can i please have the video implementation of the object detection method. The challenge i am facing is of the model using up all my resources for inference and i am sure this method goes a long way in ensuring efficient resource usage during inference.

Adrian Rosebrock September 11, 2017 at 4:08 pm #

REPLY

I will be sharing the video implementation of the deep learning object detection algorithm on Monday, September 18th. Be sure to keep an eye on your inbox as I'll be announcing the tutorial via email.

Sydney September 12, 2017 at 3:40 am #

REPLY

Thanks a lot man

Terry September 11, 2017 at 6:23 pm #

REPLY

God send you to save my life. I struggled for months about the performance issue with yolov2. It's just too heavy for cpu and mobile devices.

Hilman September 11, 2017 at 6:35 pm #

REPLY

Adrian, I am glad there is someone like you in this CV/ML community!
Keep up the high quality contents!

Adrian Rosebrock September 12, 2017 at 7:18 am #

REPLY

Thanks Hilman!

Chris Albertson September 11, 2017 at 6:36 pm #

REPLY

I'm still trying to understand how an image classifier cold be incorporated into a larger network for find bounding boxes. I thought about searching a tree of cropped images buy that would be interactive and slow.

I looks like this article took the black-box approach. How to detect objects? Make a call to an object detector. That's easy but how does the object detector work?

How can an object classifier like vgg16 be used for deception without iteration

Adrian Rosebrock September 12, 2017 at 7:18 am #

REPLY

Traditional object detection is accomplished using a sliding window an image pyramid, like in Histogram of Oriented Gradients. Deep learning-based object detectors do end-to-end object detection. The actual inner workings of how SSD/Faster R-CNN work are outside the context of this post, but the gist is that you can divide an image into a grid, classify each grid, and then adjust the anchors of the grid to better fit the object. This is a huge simplification but it should help point you in the right direction.

Barbara September 11, 2017 at 7:39 pm #

REPLY

Hi Adrian, how can I edit your code to only detect person? The others shapes aren't necessary for me. And thank you so much for your tutorial, it helps a lot

Adrian Rosebrock September 12, 2017 at 7:16 am #

REPLY

The "person" class is the 14th index in `CLASSES` and therefore the returned `detections` as well. You can remove the `for` loop that loops over the detections and then just check the probability associated with the person class:

```
confidence = detections[0, 0, 14, 2]
if confidence > threshold:
    ...
```

Barbara September 12, 2017 at 12:29 pm #

REPLY

Thank you so much. You have no idea of how much your tutorials help

Barbara September 12, 2017 at 1:01 pm #

REPLY

It didn't work. the detections return only the shapes that were detected. if I had only 2 shapes in my image, the for loop will repeat twice, then integration would be 0 and 1 and not the whole CLASSES. So, your answer is wrong. I've tried it. But I can't find a way of detecting only human shape.

Adrian Rosebrock September 12, 2017 at 2:06 pm #

REPLY

Try this:

```
for i in np.arange(0, detections.shape[2]):
    confidence = detections[0, 0, i, 2]
    idx = int(detections[0, 0, i, 1])

    if idx == 14 and confidence > threshold:
        print("person")
```

You'll want to double-check that the `idx` is indeed 14.

Barbara September 12, 2017 at 2:40 pm #

That's is exactly what I tried, but it's 15 for "person". You said in other comment that you'd be sharing the video implementation on Monday. I already did that following the instructions here and others about video. But, it takes around 17 s between frames (between processing a frame and another). Do you know what I could do to decrease this time?

Adrian Rosebrock September 12, 2017 at 6:05 pm #

Hi Barbara — unfortunately without knowing more about your setup I'm not sure what the issue is. I would kindly ask you to please wait until the video tutorial is released on Monday, September 18th. There are additional optimizations that you may not be considering such as reducing frame size, using threading to speedup the frames per second rate, etc.

siam September 12, 2017 at 3:12 am #

REPLY

after running that code i found that error:argument -i/–image is required
How can I fix it?
I am using windows 10, and python 2.7

Adrian Rosebrock September 12, 2017 at 7:14 am #

REPLY

Hi Siam — you are not providing the `--image` command line argument. Please (1) see my examples of executing the script in this tutorial and (2) read up on command line arguments.

Alexander September 12, 2017 at 7:12 am #

REPLY

Thank you, Adrian. Very useful theme with interest explanation.

Adrian Rosebrock September 12, 2017 at 7:19 am #

REPLY

I'm happy you found it helpful, Alexander! It's my pleasure to share.

Jose fernando September 12, 2017 at 1:09 pm #

REPLY

hello adrian I am from Colombia you would recommend using linux for a higher performance or no problem if you use windows Thanks

Adrian Rosebrock September 12, 2017 at 2:06 pm #

REPLY

I would definitely recommend using Linux for deep learning environments. macOS is a good fallback or if you're just playing around and learning fundamentals. I would not recommend Windows.

Thimira Amaratunga September 13, 2017 at 12:16 pm #

REPLY

Hi Adrian,

Is it possible to use a pre-trained TensorFlow model with OpenCV 3.3 as a custom object detector? Or does it only work with Caffe?

Thanks,

Adrian Rosebrock September 13, 2017 at 2:53 pm #

REPLY

You can use a pre-trained TensorFlow model. Please see my reply to "Sydney".

Walid Ahmed September 13, 2017 at 1:41 pm #

REPLY

Thanks a lot

your simple illustration for complex new issues is highly appreciated,

Adrian Rosebrock September 13, 2017 at 2:52 pm #

REPLY

Thanks Walid, I'm happy that you enjoyed the tutorial! 🙂

Sydney September 13, 2017 at 2:21 pm #

REPLY

Hie man. How can i use a tensorflow .pb model file instead of he caffee model?

Adrian Rosebrock September 13, 2017 at 2:52 pm #

REPLY

Please see this blog post where I list out the TensorFlow functions for OpenCV.

Flávio Rodrigues September 13, 2017 at 3:25 pm #

REPLY

Hi, Adrian. Have you tried the original TensorFlow Model to compare with the Caffe version? Do you plan to do such tests and show on your blog how to use a pre-trained model with differentt Network architectures? Thanks a lot for your great posts. It encourages me even more to buy your books, and I hope I will!

Adrian Rosebrock September 13, 2017 at 3:35 pm #

REPLY

I personally haven't benchmarked the original TensorFlow model compared to the Caffe one; however, the author of the TensorFlow did benchmark them. They share their benchmarks here and note the differences in implementation.

I've already covered how to use GoogLeNet and now MobileNet in this post. I'll cover more networks in the future. Otherwise, for a detailed review of other state-of-the-art architectures (and how to implement them) I would definitely refer you to Deep Learning for Computer Vision with Python.

Flávio Rodrigues September 13, 2017 at 3:54 pm #

REPLY

Thanks a lot, Adrian. And I have just watched your new real-time object detection video on YouTube. Oh, man, stop blowing my mind! Hahaha. I can't wait to see the blog post. And thank you for always answering our questions. You must be a super organized person to do that on such a busy schedule. Cheers.

Adrian Rosebrock September 14, 2017 at 6:33 am #

REPLY

Thanks Flávio, it's my pleasure to help

Alan Federman September 14, 2017 at 12:54 pm #

REPLY

Traceback (most recent call last):
File "deep_learning_object_detection.py", line 32, in
net = cv2.dnn.readNetFromCaffe(args["prototxt"], args["model"])
AttributeError: 'module' object has no attribute 'dnn'

I missed a step somewhere.

Adrian Rosebrock September 14, 2017 at 1:13 pm #

REPLY

Hi Alan — it looks like you do not have OpenCV 3.3 installed. Please ensure OpenCV 3.3 has been installed on your system.

Leave a Reply

Name (required)

Email (will not be published) (required)

Website

SUBMIT COMMENT