PYIMAGESEARCH

DEEP LEARNING (HTTPS://WWW.PYIMAGESEARCH.COM/CATEGORY/DEEP-LEARNING/)

FACE APPLICATIONS (HTTPS://WWW.PYIMAGESEARCH.COM/CATEGORY/FACES/)

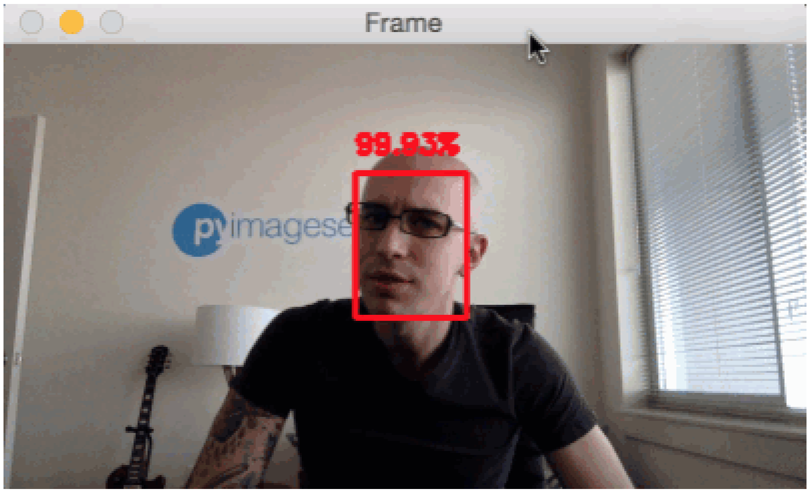
OPENCV TUTORIALS (HTTPS://WWW.PYIMAGESEARCH.COM/CATEGORY/OPENCV/)

TUTORIALS (HTTPS://WWW.PYIMAGESEARCH.COM/CATEGORY/TUTORIALS/)

Face detection with OpenCV and deep learning

Click here to download the source code to this post

by Adrian Rosebrock (https://www.pyimagesearch.com/author/adrian/) *on* February 26, 2018



(https://pyimagesearch.com/wp-content/uploads/2018/02/deep_learning_face_detection_opencv.gif)

Today I'm going to share a little known secret with you regarding the OpenCV library:

used for face detection…

…but I’m willing to bet that you don’t know about the **“hidden” deep learning-based face detector** that has been part of OpenCV since OpenCV 3.3.

In the remainder of today’s blog post I’ll discuss:

- Where this “hidden” deep learning face detector lives in the OpenCV library
- How you can perform **face detection in *images*** using OpenCV and deep learning
- How you can perform **face detection in *video*** using OpenCV and deep learning

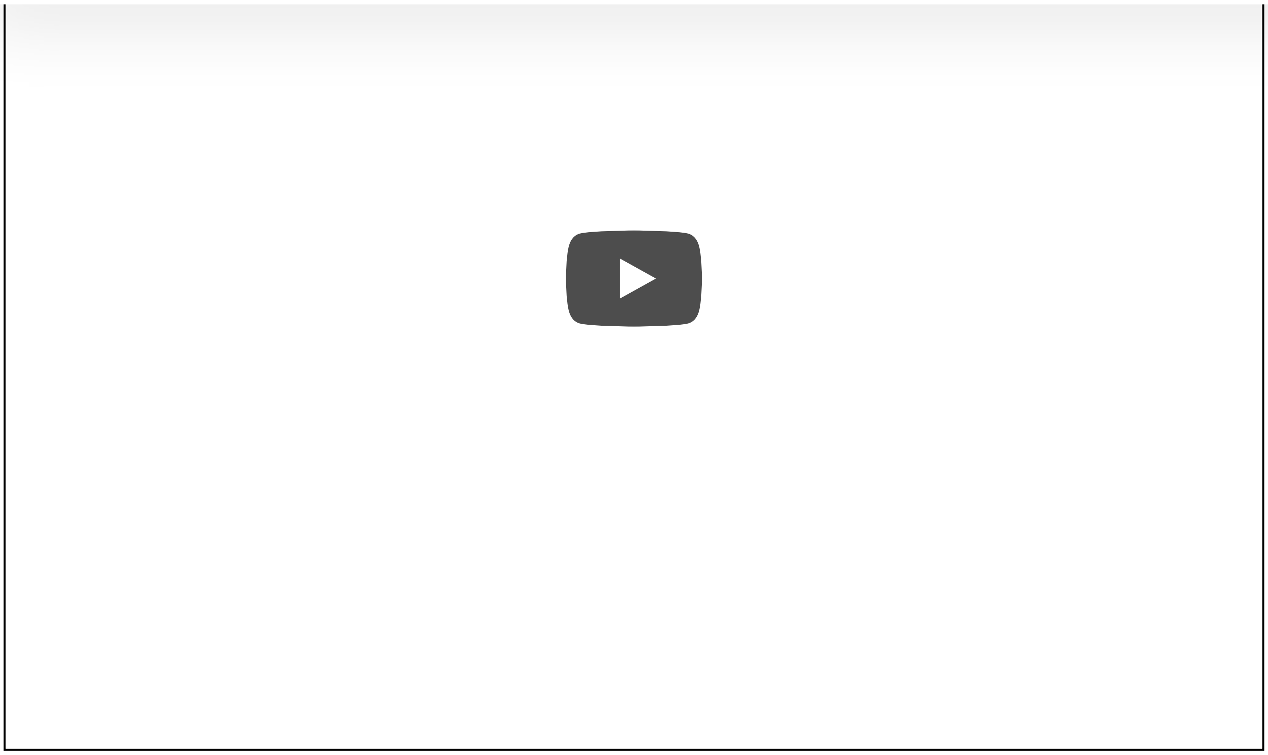
As we’ll see, it’s easy to swap out Haar cascades for their more accurate deep learning face detector counterparts.

To learn more about face detection with OpenCV and deep learning, *just keep reading!*

Looking for the source code to this post?

JUMP RIGHT TO THE DOWNLOADS SECTION →

Face detection with OpenCV and deep learning



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

In the first part we’ll discuss the origin of the more accurate OpenCV face detectors and where they live inside the OpenCV library.

From there I’ll demonstrate how you can perform face detection in images using OpenCV and deep learning.

I’ll then wrap up the blog post discussing how you can apply face detection to video streams using OpenCV and deep learning as well.

Where do these “better” face detectors live in OpenCV and where did they come from?

Back in August 2017, **OpenCV 3.3 was officially released**

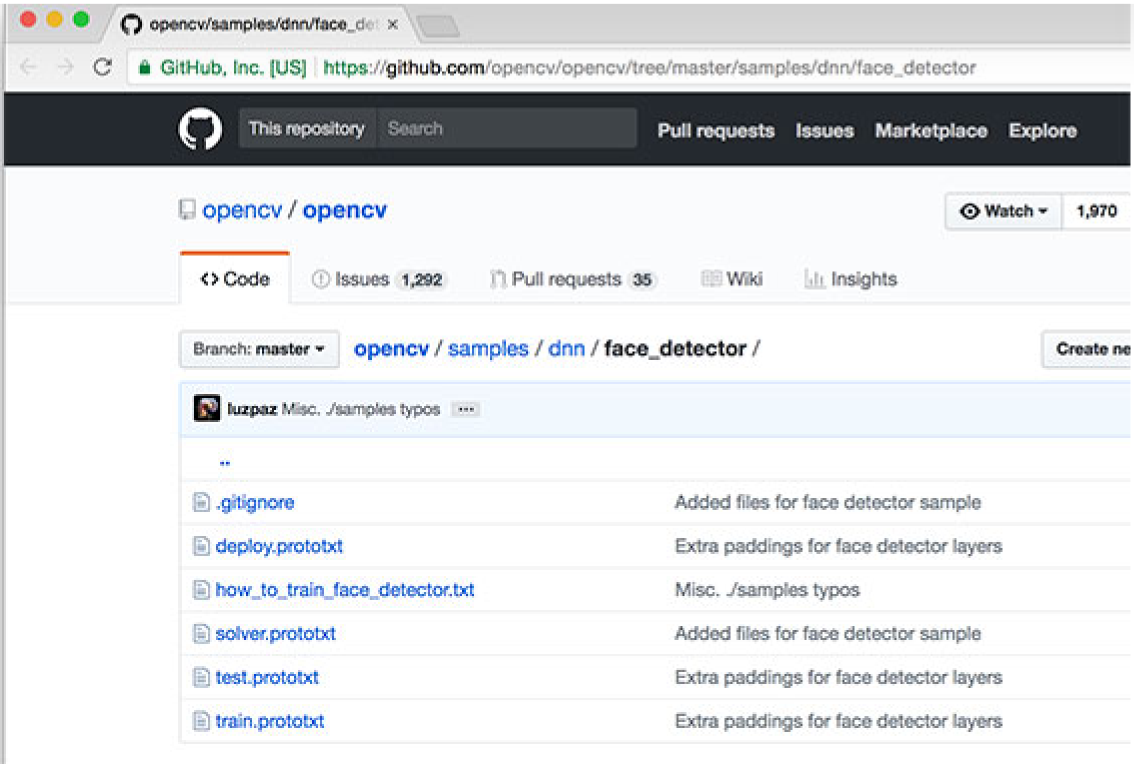
This module supports a number of deep learning frameworks, including Caffe, TensorFlow, and Torch/PyTorch.

The primary contributor to the `dnn` module, **Aleksandr Rybnikov (https://github.com/arrybn)**, has put a *huge* amount of work into making this module possible (and we owe him a big round of thanks and applause).

And since the release of OpenCV 3.3, I've been sharing a number of deep learning OpenCV tutorials, including:

- ***Deep Learning with OpenCV (https://pyimagesearch.com/2017/08/21/deep-learning-with-opencv/)***
- ***Object detection with deep learning and OpenCV (https://pyimagesearch.com/2017/09/11/object-detection-with-deep-learning-and-opencv/)***
- ***Real-time object detection with deep learning and OpenCV (https://pyimagesearch.com/2017/09/18/real-time-object-detection-with-deep-learning-and-opencv/)***
- ***Deep learning on the Raspberry Pi with OpenCV (https://pyimagesearch.com/2017/10/02/deep-learning-on-the-raspberry-pi-with-opencv/)***
- ***Raspberry Pi: Deep learning object detection with OpenCV (https://pyimagesearch.com/2017/10/16/raspberry-pi-deep-learning-object-detection-with-opencv/)***
- ***Deep learning: How OpenCV's blobFromImage works (https://pyimagesearch.com/2017/11/06/deep-learning-opencvs-blobfromimage-works/)***

However, what most OpenCV users do not know is that Rybnikov has included a more accurate, deep learning-based face detector included in the official release of OpenCV (although it can be a bit hard to find if you don't know where to look).



https://github.com/opencv/opencv/tree/master/samples/dnn/face_detector

opencv / opencv

opencv / samples / dnn / face_detector /

luzpaz Misc. ./samples typos …

..	
.gitignore	Added files for face detector sample
deploy.prototxt	Extra paddings for face detector layers
how_to_train_face_detector.txt	Misc. ./samples typos
solver.prototxt	Added files for face detector sample
test.prototxt	Extra paddings for face detector layers
train.prototxt	Extra paddings for face detector layers

(https://pyimagesearch.com/wp-content/uploads/2018/02/deep_learning_face_detection_repo.jpg)

Figure 1: The OpenCV repository on GitHub has an example of deep learning face detection.

When using OpenCV's deep neural network module with Caffe models, you'll need two sets of files:

- The **.prototxt** file(s) which define the *model architecture* (i.e., the layers themselves)
- The **.caffemodel** file which contains the *weights* for the actual layers

Both files are required when using models trained using Caffe for deep learning.

However, you'll only find the prototxt files here in the GitHub repo.

The weight files are *not* included in the OpenCV `samples` directory and it requires a bit more digging to find them…

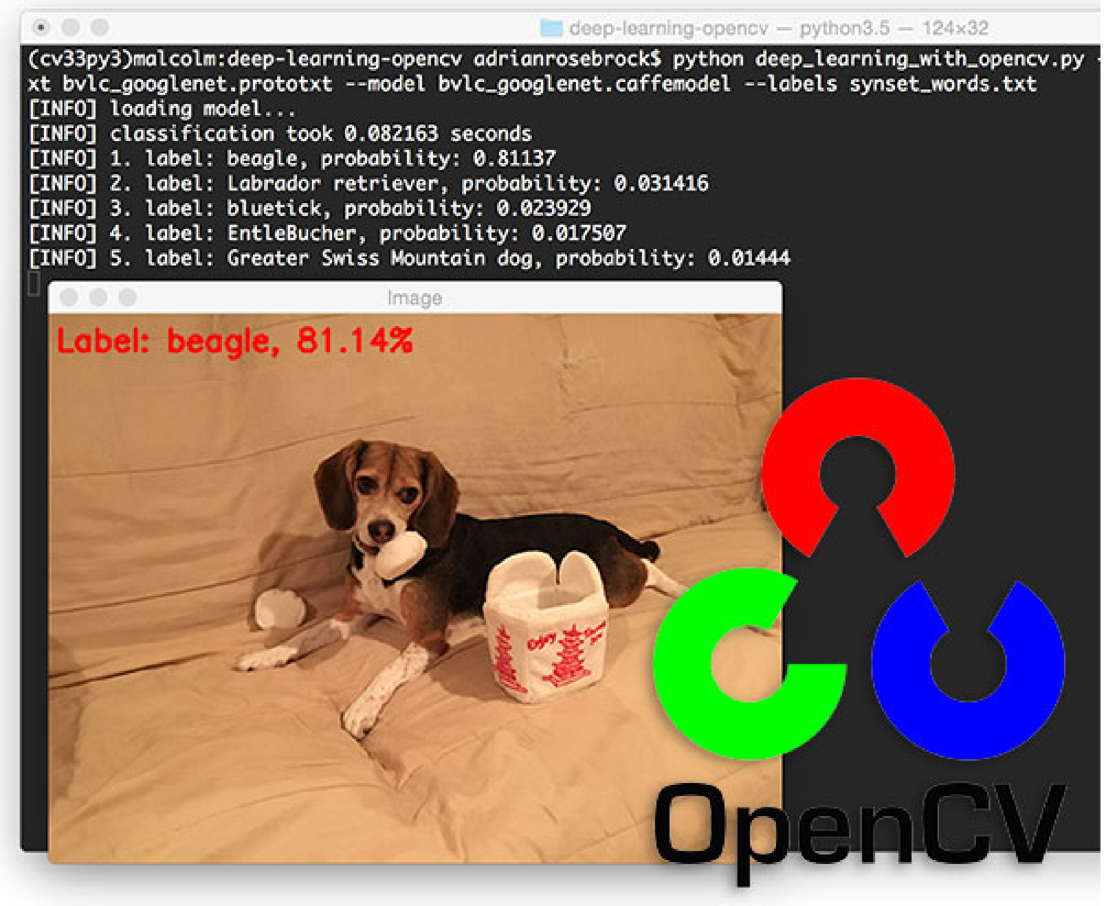
For your convenience, I have included *both* the:

1. Caffe prototxt files
2. and Caffe model weight files

...inside the *"Downloads"* section of this blog post.

To skip to the downloads section, just click here.

How does the OpenCV deep learning face detector work?



```
(cv33py3)malcolm:deep-learning-opencv adrianrosebrock$ python deep_learning_with_opencv.py -
xt bvlc_googlenet.prototxt --model bvlc_googlenet.caffemodel --labels synset_words.txt
[INFO] loading model...
[INFO] classification took 0.082163 seconds
[INFO] 1. label: beagle, probability: 0.81137
[INFO] 2. label: Labrador retriever, probability: 0.031416
[INFO] 3. label: bluetick, probability: 0.023929
[INFO] 4. label: EntleBucher, probability: 0.017507
[INFO] 5. label: Greater Swiss Mountain dog, probability: 0.01444
```

ResNet base network (unlike other OpenCV SSDs that you may have seen which typically use MobileNet as the base network).

A full review of SSDs and ResNet is outside the scope of this blog post, so if you're interested in learning more about Single Shot Detectors (including how to train your own custom deep learning object detectors), start with this article **here on the PyImageSearch blog (https://pyimagesearch.com/2017/09/11/object-detection-with-deep-learning-and-opencv/)** and then take a look at my book, ***Deep Learning for Computer Vision with Python (https://pyimagesearch.com/deep-learning-computer-vision-python-book/)***, which includes in-depth discussions and code enabling you to train your own object detectors.

Face detection in images with OpenCV and deep learning

In this first example we'll learn how to apply face detection with OpenCV to single input images.

In the next section we'll learn how to modify this code and apply face detection with OpenCV to videos, video streams, and webcams.

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

Face detection with OpenCV and deep learning

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

# construct the argument parse and parse the arguments
ap = argparse.ArgumentParser()
ap.add_argument("-i", "--image", required=True,
   help="path to input image")
ap.add_argument("-p", "--prototxt", required=True,
   help="path to Caffe 'deploy' prototxt file")
ap.add_argument("-m", "--model", required=True,
   help="path to Caffe pre-trained model")
ap.add_argument("-c", "--confidence", type=float, default=0.5,
   help="minimum probability to filter weak detections")
args = vars(ap.parse_args())
```

- `--image` : The path to the input image.
- `--prototxt` : The path to the Caffe prototxt file.
- `--model` : The path to the pretrained Caffe model.

An optional argument, `--confidence` , can overwrite the default threshold of 0.5 if you wish.

From there lets load our model and create a blob from our image:

Face detection with OpenCV and deep learning

```
18. | # load our serialized model from disk
19. | print("[INFO] loading model...")
20. | net = cv2.dnn.readNetFromCaffe(args["prototxt"], args["model"])
21. |
22. | # load the input image and construct an input blob for the image
23. | # by resizing to a fixed 300x300 pixels and then normalizing it
24. | image = cv2.imread(args["image"])
25. | (h, w) = image.shape[:2]
26. | blob = cv2.dnn.blobFromImage(cv2.resize(image, (300, 300)), 1.0,
27. |     (300, 300), (104.0, 177.0, 123.0))
```

First, we load our model using our `--prototxt` and `--model` file paths. We store the model as `net` (**Line 20**).

Then we load the `image` (**Line 24**), extract the dimensions (**Line 25**), and create a `blob` (**Lines 26 and 27**).

The `dnn.blobFromImage` takes care of pre-processing which includes setting the `blob` dimensions and normalization. If you're interested in learning more about the `dnn.blobFromImage` function, I review in detail in **this blog post (https://pyimagesearch.com/2017/11/06/deep-learning-opencvs-blobfromimage-works/)**.

Next, we'll apply face detection:

Face detection with OpenCV and deep learning

```
29. | # pass the blob through the network and obtain the detections and
30. | # predictions
31. | print("[INFO] computing object detections...")
32. | net.setInput(blob)
33. | detections = net.forward()
```

Face detection with OpenCV and deep learning

```
35. |  # loop over the detections
36. |  for i in range(0, detections.shape[2]):
37. |     # extract the confidence (i.e., probability) associated with the
38. |     # prediction
39. |     confidence = detections[0, 0, i, 2]
40. |
41. |     # filter out weak detections by ensuring the `confidence` is
42. |     # greater than the minimum confidence
43. |     if confidence > args["confidence"]:
44. |         # compute the (x, y)-coordinates of the bounding box for the
45. |         # object
46. |         box = detections[0, 0, i, 3:7] * np.array([w, h, w, h])
47. |         (startX, startY, endX, endY) = box.astype("int")
48. |
49. |         # draw the bounding box of the face along with the associated
50. |         # probability
51. |         text = "{:.2f}%".format(confidence * 100)
52. |         y = startY - 10 if startY - 10 > 10 else startY + 10
53. |         cv2.rectangle(image, (startX, startY), (endX, endY),
54. |             (0, 0, 255), 2)
55. |         cv2.putText(image, text, (startX, y),
56. |             cv2.FONT_HERSHEY_SIMPLEX, 0.45, (0, 0, 255), 2)
57. |
58. |  # show the output image
59. |  cv2.imshow("Output", image)
60. |  cv2.waitKey(0)
```

We begin looping over the detections on **Line 36**.

From there, we extract the `confidence` (**Line 39**) and compare it to the confidence threshold (**Line 43**). We perform this check to filter out weak detections.

If the confidence meets the minimum threshold, we proceed to draw a rectangle and along with the *probability* of the detection on **Lines 46-56**.

To accomplish this, we first calculate the *(x, y)*-coordinates of the bounding box (**Lines 46 and 47**).

We then build our confidence `text` string (**Line 51**) which contains the probability of the detection.

In case the our `text` would go off-image (such as when the face detection occurs at the very top of an image), we shift it down by 10 pixels (**Line 52**).

Our face rectangle and confidence `text` is drawn on the `image` on **Lines 53-56**.

Let’s try out the OpenCV deep learning face detector.

Make sure you use the *“Downloads”* section of this blog post to download:

- The **source code** used in this blog post
- The **Caffe prototxt files** for deep learning face detection
- The **Caffe weight files** used for deep learning face detection
- The **example images** used in this post

From there, open up a terminal and execute the following command:

Face detection with OpenCV and deep learning

```
$ python detect_faces.py --image rooster.jpg --prototxt deploy.prototxt.txt \
    --model res10_300x300_ssd_iter_140000.caffemodel
```

(https://pyimagesearch.com/wp-content/uploads/2018/02/deep_learning_face_detection_example_01.jpg)

Figure 3: My face is detected in this image with 74% confidence using the OpenCV deep learning face detector.

The above photo is of me during my first trip to Ybor City in Florida, where chickens are allowed to roam free throughout the city. There are even laws protecting the chickens which I thought was very cool. Even though I grew up in rural farmland, I was still totally surprised to see a rooster crossing the road — which of course spawned many *“Why did the chicken cross the road?”* jokes.

Here you can see my face is detected with 74.30% confidence, even though my face is at an angle. OpenCV’s Haar cascades are notorious for missing faces that are not at a “straight on” angle, but by using OpenCV’s deep learning face detectors, we are able to detect my face.

And now we’ll see how another example works, this time with three faces:

```
Face detection with OpenCV and deep learning
```

(https://pyimagesearch.com/wp-content/uploads/2018/02/deep_learning_face_detection_example_02.jpg)

Figure 4: The OpenCV DNN face detector finds all three images without any trouble.

This photo was taken in Gainesville, FL after one of my favorite bands finished up a show at Loosey’s, a popular bar and music venue in the area. Here you can see my fiancée (*left*), me (*middle*), and Jason (*right*), a member of the band.

detectors are over their standard Haar cascade counterparts shipped with the library.

Face detection in video and webcam with OpenCV and deep learning

Now that we have learned how to apply face detection with OpenCV to single images, let's also apply face detection to videos, video streams, and webcams.

Luckily for us, most of our code in the previous section on face detection with OpenCV in single images can be reused here!

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

Face detection with OpenCV and deep learning

```python
# import the necessary packages
from imutils.video import VideoStream
import numpy as np
import argparse
import imutils
import time
import cv2

# construct the argument parse and parse the arguments
ap = argparse.ArgumentParser()
ap.add_argument("-p", "--prototxt", required=True,
   help="path to Caffe 'deploy' prototxt file")
ap.add_argument("-m", "--model", required=True,
   help="path to Caffe pre-trained model")
ap.add_argument("-c", "--confidence", type=float, default=0.5,
   help="minimum probability to filter weak detections")
args = vars(ap.parse_args())
```

Compared to above, we will need to import three additional packages: `VideoStream` , `imutils` , and `time` .

If you don't have `imutils` in your virtual environment, you can install it via:

Face detection with OpenCV and deep learning

```
$ pip install imutils
```

Face detection with OpenCV and deep learning

```
19. | # load our serialized model from disk
20. | print("[INFO] loading model...")
21. | net = cv2.dnn.readNetFromCaffe(args["prototxt"], args["model"])
22. |
23. | # initialize the video stream and allow the camera sensor to warm up
24. | print("[INFO] starting video stream...")
25. | vs = VideoStream(src=0).start()
26. | time.sleep(2.0)
```

Loading the model is the same as above.

We initialize a `VideoStream` object specifying camera with index zero as the source (in general this would be your laptop's built in camera or your desktop's first camera detected).

A few quick notes here:

- **Raspberry Pi + picamera users** can replace **Line 25** with `vs = VideoStream(usePiCamera=True).start()` if you wish to use the Raspberry Pi camera module.
- If you to parse a **video file** (rather than a video stream) swap out the `VideoStream` class for `FileVideoStream`. You can learn more about the **FileVideoStream class in this blog post (https://pyimagesearch.com/2017/02/06/faster-video-file-fps-with-cv2-videocapture-and-opencv/)**.

We then allow the camera sensor to warm up for 2 seconds (**Line 26**).

From there we loop over the frames and compute face detections with OpenCV:

Face detection with OpenCV and deep learning

```
28. | # loop over the frames from the video stream
29. | while True:
30. |     # grab the frame from the threaded video stream and resize it
31. |     # to have a maximum width of 400 pixels
32. |     frame = vs.read()
33. |     frame = imutils.resize(frame, width=400)
34. |
35. |     # grab the frame dimensions and convert it to a blob
36. |     (h, w) = frame.shape[:2]
37. |     blob = cv2.dnn.blobFromImage(cv2.resize(frame, (300, 300)), 1.0,
38. |         (300, 300), (104.0, 177.0, 123.0))
```

This block should look mostly familiar to the static image version in the previous section.

In this block, we're reading a `frame` from the video stream (**Line 32**), creating a `blob` (**Lines 37 and 38**), and passing the `blob` through the deep neural `net` to obtain face detections (**Lines 42 and 43**).

We can now loop over the detections, compare to the confidence threshold, and draw face boxes + confidence values on the screen:

Face detection with OpenCV and deep learning

```
45.     # loop over the detections
46.     for i in range(0, detections.shape[2]):
47.         # extract the confidence (i.e., probability) associated with the
48.         # prediction
49.         confidence = detections[0, 0, i, 2]
50.
51.         # filter out weak detections by ensuring the `confidence` is
52.         # greater than the minimum confidence
53.         if confidence < args["confidence"]:
54.             continue
55.
56.         # compute the (x, y)-coordinates of the bounding box for the
57.         # object
58.         box = detections[0, 0, i, 3:7] * np.array([w, h, w, h])
59.         (startX, startY, endX, endY) = box.astype("int")
60.
61.         # draw the bounding box of the face along with the associated
62.         # probability
63.         text = "{:.2f}%".format(confidence * 100)
64.         y = startY - 10 if startY - 10 > 10 else startY + 10
65.         cv2.rectangle(frame, (startX, startY), (endX, endY),
66.             (0, 0, 255), 2)
67.         cv2.putText(frame, text, (startX, y),
68.             cv2.FONT_HERSHEY_SIMPLEX, 0.45, (0, 0, 255), 2)
```

For a detailed review of this code block, please review the previous section where we perform face detection to still, static images. The code here is nearly identical.

Now that our OpenCV face detections have been drawn, let's display the frame on the screen and wait for a keypress:

Face detection with OpenCV and deep learning

```
70.     # show the output frame
71.     cv2.imshow("Frame", frame)
72.     key = cv2.waitKey(1) & 0xFF
```

```
cv2.destroyAllWindows()
vs.stop()
```

We display the `frame` on the screen until the “q” key is pressed at which point we `break` out of the loop and perform cleanup.

Face detection in video and webcam with OpenCV results

To try out the OpenCV deep learning face detector make sure you use the ***“Downloads”*** section of this blog post to grab:

- The **source code** used in this blog post
- The **Caffe prototxt files** for deep learning face detection
- The **Caffe weight files** used for deep learning face detection

Once you have downloaded the files, running the deep learning OpenCV face detector with a webcam feed is easy with this simple command:

Face detection with OpenCV and deep learning

```
$ python detect_faces_video.py --prototxt deploy.prototxt.txt \
    --model res10_300x300_ssd_iter_140000.caffemodel
```

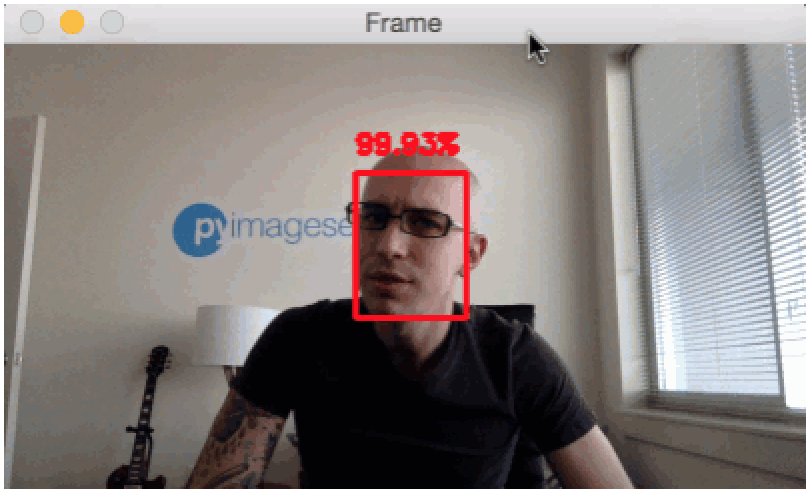
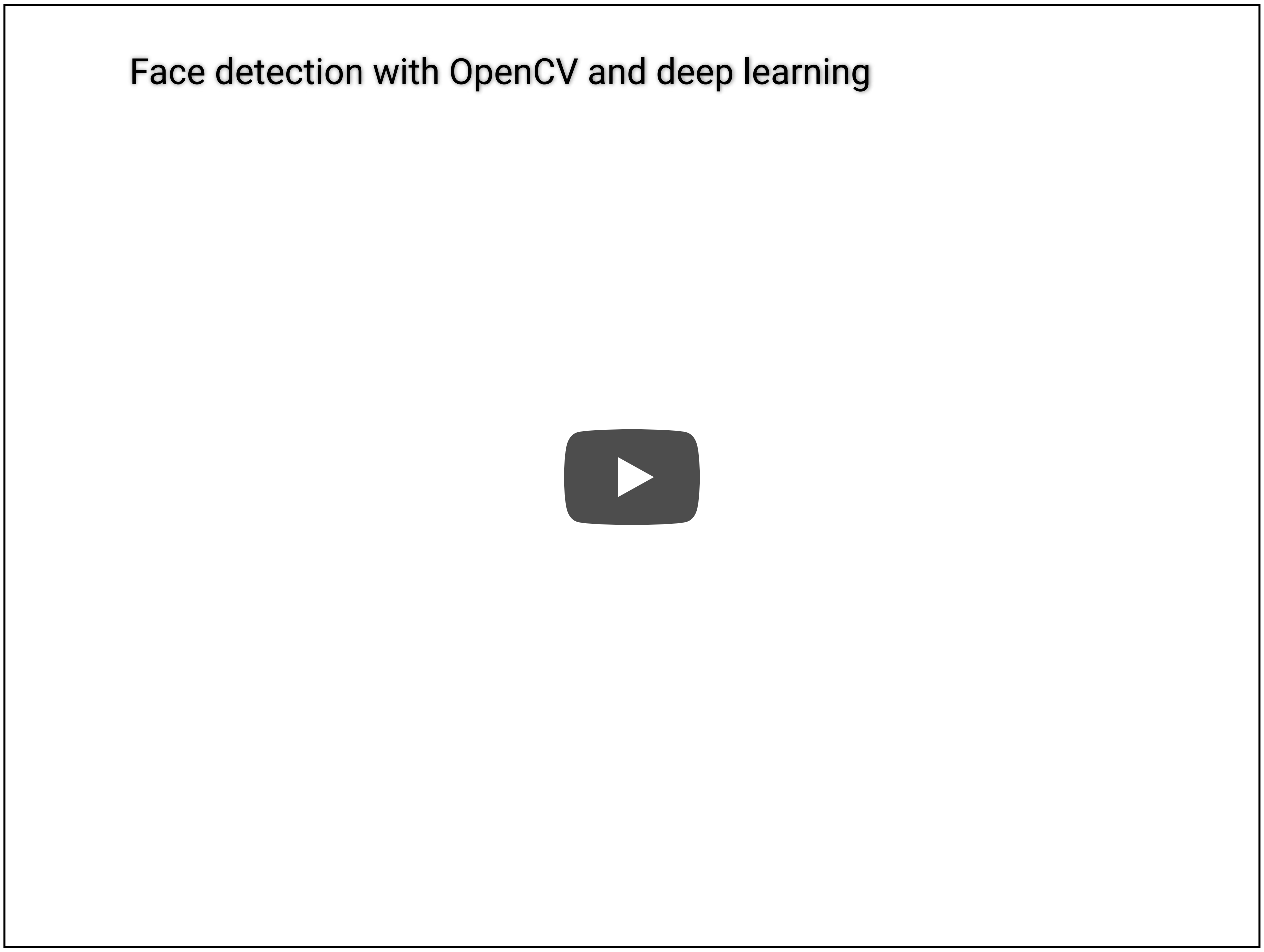


Figure 3: Face detection in video with OpenCV's DNN module.

You can see a full video demonstration, including my commentary, in the following video:



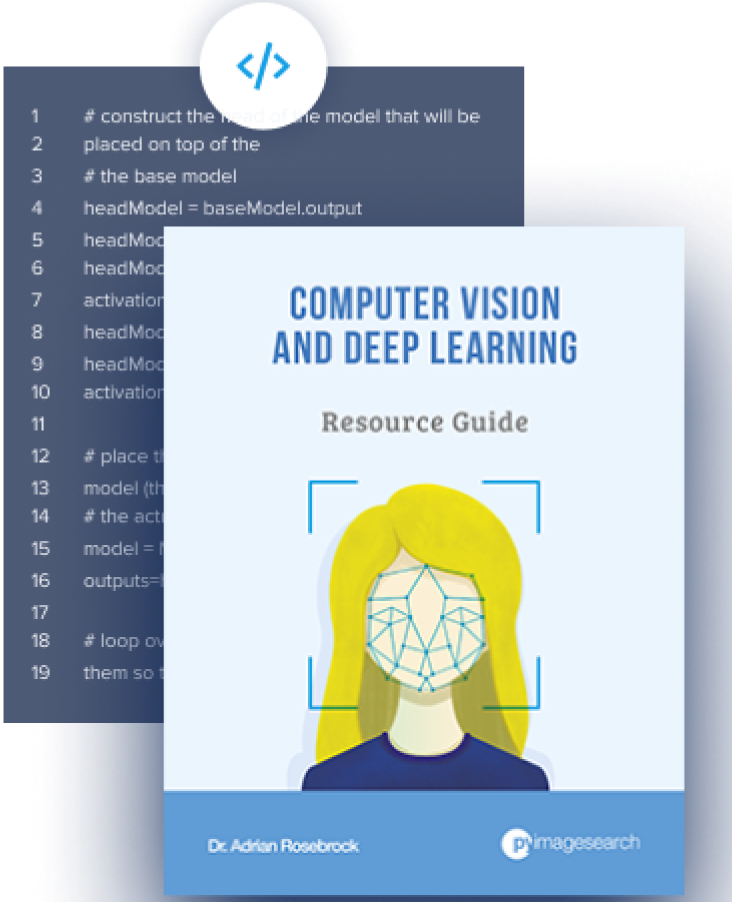
Summary

In today's blog post you discovered a little known secret about the OpenCV library — **OpenCV ships out-of-the-box with a more accurate face detector** (as compared to OpenCV's Haar cascades).

The more accurate OpenCV face detector is **deep learning based**, and in particular, utilizes the Single Shot Detector (SSD) framework with ResNet as the base network.

for your convenience, I have gathered the Caffe prototxt and weight files for you — **just use the *"Downloads"* form below to download the (more accurate) deep learning-based OpenCV face detector.**

See you next week with another great computer vision + deep learning tutorial!



```
# construct the head of the model that will be
placed on top of the
# the base model
headModel = baseModel.output
headMod
headMod
activation
headMod
headMod
activation

# place t
model (th
# the acti
model = M
outputs=
 
# loop ov
them so t
```

Download the Source Code and FREE 17-page Resource Guide