

[Click here to download the source code to this post](#)

PYIMAGESEARCH

DEEP LEARNING ([HTTPS://WWW.PYIMAGESEARCH.COM/CATEGORY/DEEP-LEARNING/](https://www.pyimagesearch.com/category/deep-learning/))

FACE APPLICATIONS ([HTTPS://WWW.PYIMAGESEARCH.COM/CATEGORY/FACES/](https://www.pyimagesearch.com/category/faces/))

OPENCV TUTORIALS ([HTTPS://WWW.PYIMAGESEARCH.COM/CATEGORY/OPENCV/](https://www.pyimagesearch.com/category/opencv/))

TUTORIALS ([HTTPS://WWW.PYIMAGESEARCH.COM/CATEGORY/TUTORIALS/](https://www.pyimagesearch.com/category/tutorials/))

Face detection with OpenCV and deep learning

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

Last updated on July 4, 2021.



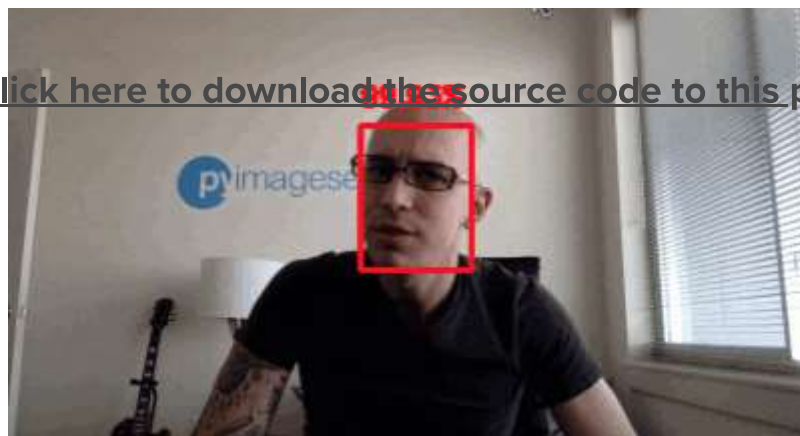


Today I'm going to share a little known secret with you regarding the OpenCV library:

You can perform *fast, accurate* face detection with OpenCV using a pre-trained deep learning face detector model shipped with the library.



[Click here to download the source code to this post](#)



You may already know that OpenCV ships out-of-the-box with pre-trained Haar cascades that can be 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:

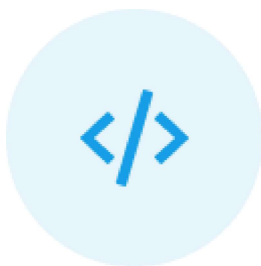
- Where this “hidden” deep learning face detector lives in the OpenCV library
- How you can perform **face detection in *images*** using OpenCV and deep learning
- How you can perform **face detection in *video*** using OpenCV and deep learning

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!*

- **Update July 2021:** Included a new section on alternative face detection methods you may want to consider for your projects.

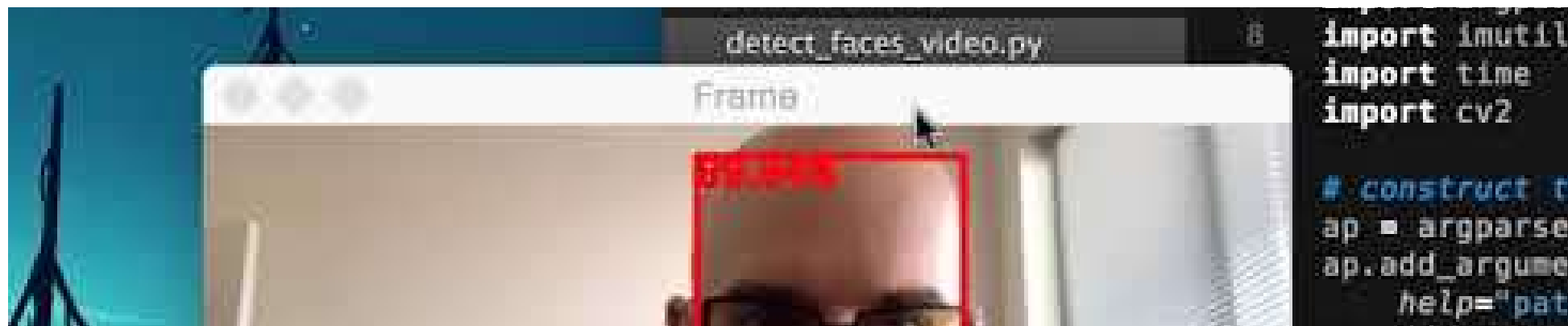
[Click here to download the source code to this post](#)

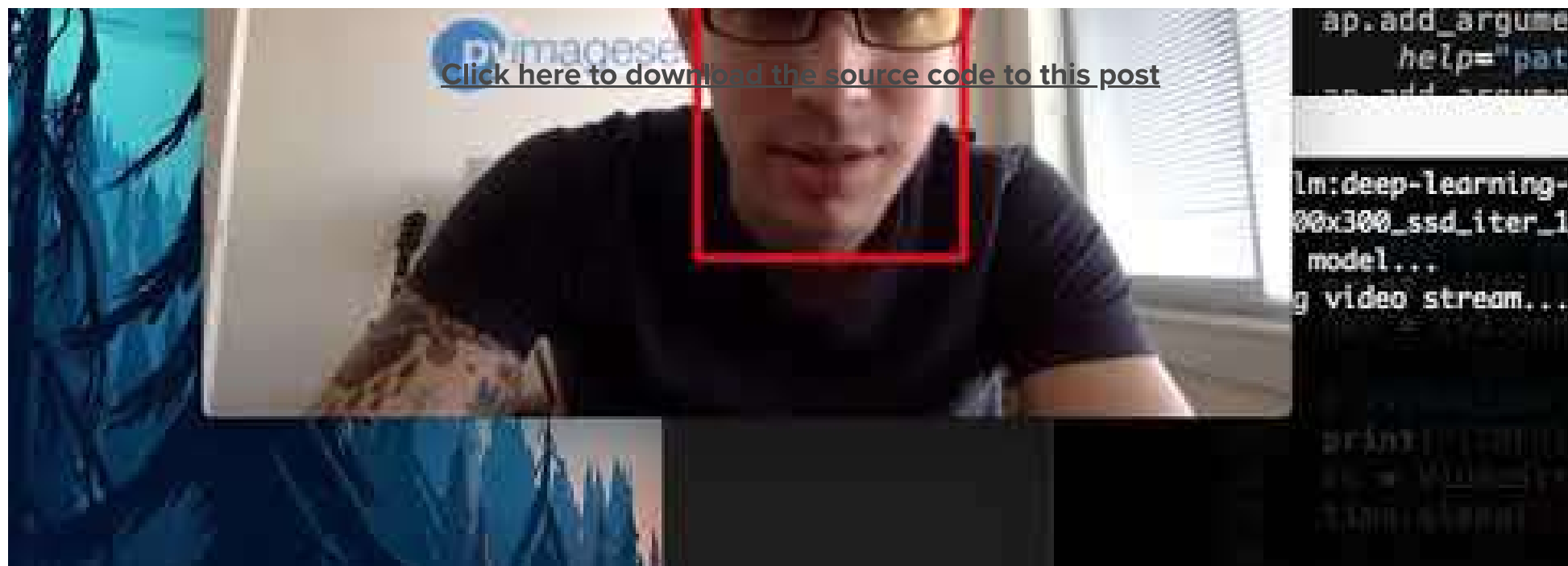


Looking for the source code to this post?

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?

[Click here to download the source code to this post](#)

Back in August 2017, **OpenCV 3.3 was officially released** (<https://pyimagesearch.com/2017/08/21/deep-learning-with-opencv/>), bringing it with it a highly improved “deep neural networks” (`dnn`) module.

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 \(https://pyimagesearch.com/2017/08/21/deep-learning-with-opencv/\)](https://pyimagesearch.com/2017/08/21/deep-learning-with-opencv/)***
- ***[Object detection with deep learning and OpenCV \(https://pyimagesearch.com/2017/09/11/object-detection-with-deep-learning-and-opencv/\)](https://pyimagesearch.com/2017/09/11/object-detection-with-deep-learning-and-opencv/)***
- ***[Real-time object detection with deep learning and OpenCV \(https://pyimagesearch.com/2017/09/18/real-time-object-detection-with-deep-learning-and-opencv/\)](https://pyimagesearch.com/2017/09/18/real-time-object-detection-with-deep-learning-and-opencv/)***
- ***[Deep learning on the Raspberry Pi with OpenCV \(https://pyimagesearch.com/2017/10/02/deep-learning-on-the-raspberry-pi-with-opencv/\)](https://pyimagesearch.com/2017/10/02/deep-learning-on-the-raspberry-pi-with-opencv/)***
- ***[<https://www.pyimagesearch.com/2018/02/26/face-detection-with-opencv-and-deep-learning/>](https://pyimagesearch.com/2017/10/16/raspberry-Raspberry Pi: Deep learning object detection with OpenCV (https://pyimagesearch.com/2017/10/16/raspberry-</i></div><div data-bbox=)***

[*pi-deep-learning-object-detection-with-opencv/*](#)

[Click here to download the source code to this post](#)

- [Deep learning: How OpenCV's blobFromImage works \(https://pyimagesearch.com/2017/11/06/deep-learning-opencvs-blobfromimage-works/\)](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).

The Caffe-based face detector can be found in the `face_detector` sub-directory of the [dnn samples](#) (https://github.com/opencv/opencv/tree/master/samples/dnn/face_detector):



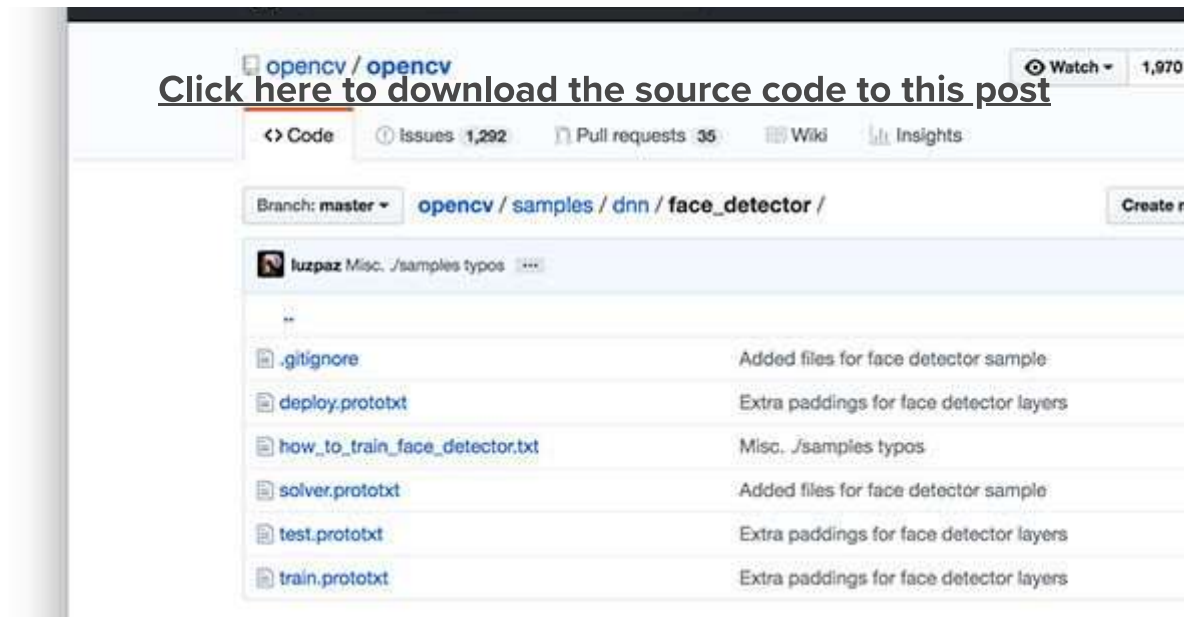


Figure 1: The OpenCV repository on GitHub has an example of deep learning face detection.

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...

Where can I get the more accurate OpenCV face detectors?

[Click here to download the source code to this post](#)

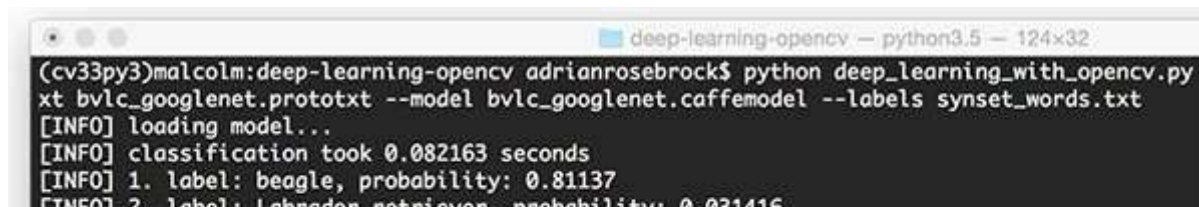
For your convenience, I have included *both* the:

- 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?



```
deep-learning-opencv — python3.5 — 124x32
(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
```

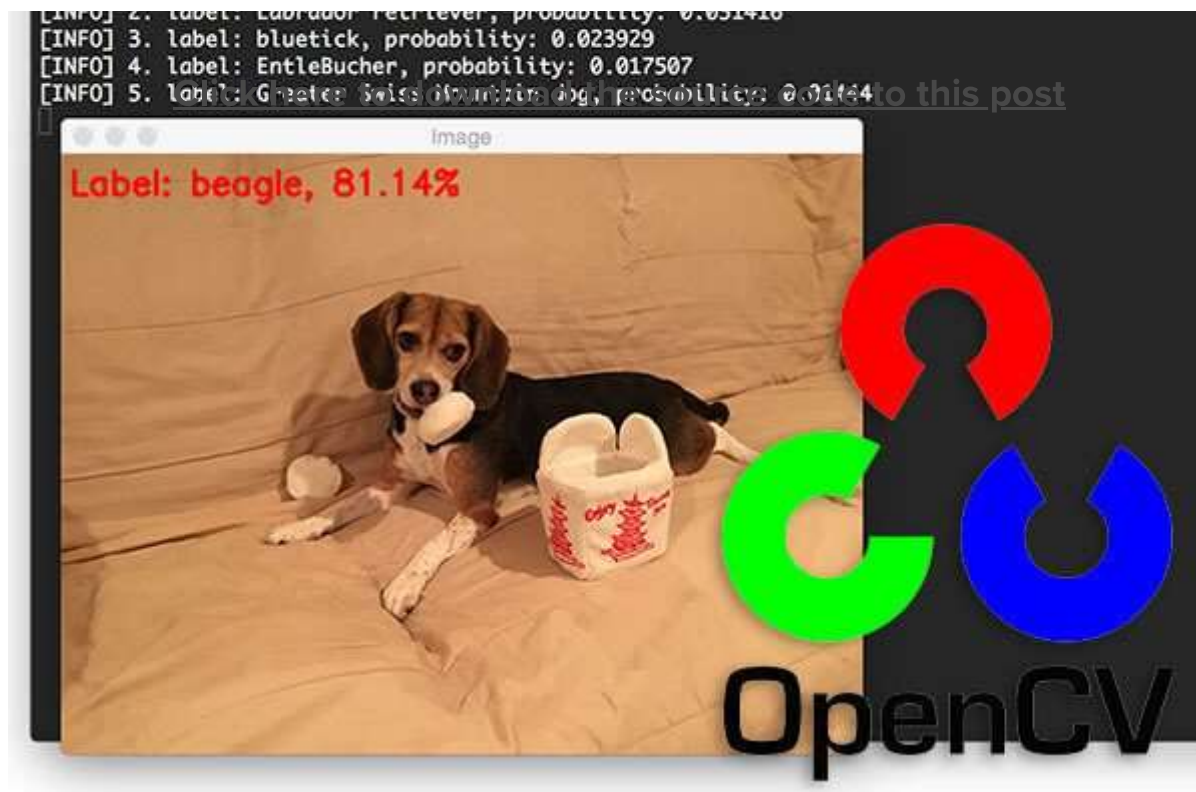


Figure 2: Deep Learning with OpenCV's DNN module.

OpenCV's deep learning face detector is based on the Single Shot Detector (SSD) framework with a 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*.

[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/) (<https://pyimagesearch.com/deep-learning-computer-vision-python-book/>), which includes in-depth discussions ~~Click here to download the source code to this post~~ 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:

→ [Launch Jupyter Notebook on Google Colab](#)

Face detection with OpenCV and deep learning

```
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 Caffee 'deploy' prototxt file")
12. | ap.add_argument("-m", "--model", required=True,
13. |     help="path to Caffee pre-trained model")
14. | ap.add_argument("-c", "--confidence", type=float, default=0.5,
15. |     help="minimum probability to filter weak detections")
16. | args = vars(ap.parse_args())
```

Here we are importing our required packages (**Lines 2-4**) and parsing command line arguments (**Lines 7-16**).

[Click here to download the source code to this post](#)

We have three required arguments:

- `--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:

→ **[Launch Jupyter Notebook on Google Colab](#)**

```

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 \(https://pyimagesearch.com/2017/11/06/deep-learning-opencvs-blobfromimage-works/\)](https://pyimagesearch.com/2017/11/06/deep-learning-opencvs-blobfromimage-works/).

Next, we'll apply face detection:

→ [Launch Jupyter Notebook on Google Colab](#)

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()
```

To detect faces, we pass the `blob` through the `net` on **Lines 32 and 33**.

And from there we'll loop over the `detections` and draw boxes around the detected faces:

→ [Launch Jupyter Notebook on Google Colab](#)

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")
```

```

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)

```

[Click here to download the source code to this post](#)

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**.

From there we loop back for additional detections following the process again. If no detections remain, we're ready to show our output image on the screen (**Lines 59 and 60**).

[Click here to download the source code to this post](#)

Face detection in images with OpenCV results

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:

→ [Launch Jupyter Notebook on Google Colab](#)

Face detection with OpenCV and deep learning

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



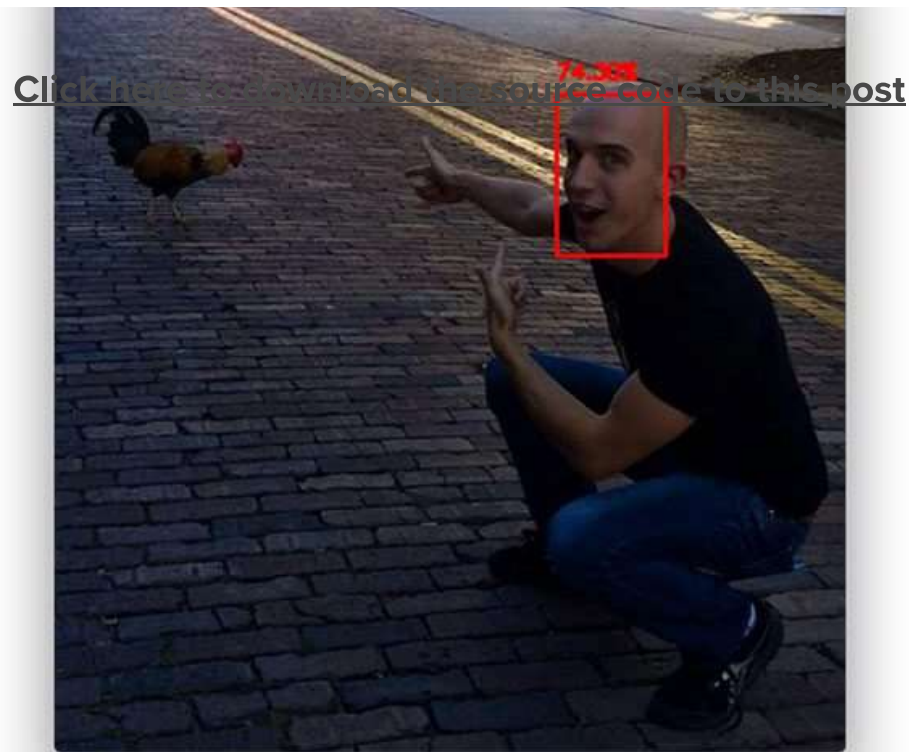


Figure 3: My face is detected in this image with 74% confidence using the OpenCV deep learning face detector.

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.

why did the chicken cross the road? jokes.

Click here to download the source code to this post

Here you can see my face is detected with 74.90% 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:

→ **[Launch Jupyter Notebook on Google Colab](#)**

Face detection with OpenCV and deep learning

```
1. | $ python detect_faces.py --image iron_chic.jpg --prototxt deploy.prototxt.txt \  
2. |    --model res10_300x300_ssd_iter_140000.caffemodel
```





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.

I'm incredibly impressed that OpenCV can detect Trisha's face despite the lighting conditions and shadows cast on her face in the dark venue (and with 86.81% probability!)
[Click here to download the source code to this post](#)

Again, this just goes to show how much better (in terms of accuracy) the deep learning OpenCV face 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:

→ [Launch Jupyter Notebook on Google Colab](#)

Face detection with OpenCV and deep learning

```
1. | # import the necessary packages
2. | from imutils.video import VideoStream
3. | import numpy as np
4. | import argparse
5. | import imutils
6. | import time
7. | import cv2
```

```

8. |
9. | # construct the argument parse and parse the arguments
10. | ap = argparse.ArgumentParser()
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("-c", "--confidence", type=float, default=0.5,
16. |     help="minimum probability to filter weak detections")
17. | args = vars(ap.parse_args())

```

[Click here to download the source code to this post](#)

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:

→ [Launch Jupyter Notebook on Google Colab](#)

Face detection with OpenCV and deep learning

```
1. | $ pip install imutils
```

Our command line arguments are mostly the same, except we do not have an `--image path` argument this time. We'll be using our webcam's video feed instead.

From there we'll load our model and initialize the video stream:

→ [Launch Jupyter Notebook on Google Colab](#)

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(args=0).start()

```

```

25. | vs = VideoStream(SRC=0).start()
26. | time.sleep(2.0)

```

[Click here to download the source code to this post](#)

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/)** (<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:

→ **[Launch Jupyter Notebook on Google Colab](#)**

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))
39. |
40. |     # pass the blob through the network and obtain the detections and
41. |     # predictions
42. |     net.setInput(blob)
43. |     detections = net.forward()

```

[Click here to download the source code to this post](#)

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:

→ [Launch Jupyter Notebook on Google Colab](#)

```

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)

```

[Click here to download the source code to this post](#)

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:

→ [Launch Jupyter Notebook on Google Colab](#)

Face detection with OpenCV and deep learning

```

70. |         # show the output frame
71. |         cv2.imshow("Frame", frame)
72. |         key = cv2.waitKey(1) & 0xFF
73. |
74. |         # if the `q` key was pressed, break from the loop
75. |         if key == ord("q"):
76. |             break
77. |
78. |         # do a bit of cleanup
79. |         cv2.destroyAllWindows()
80. |         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

[Click here to download the source code to this post](#)

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

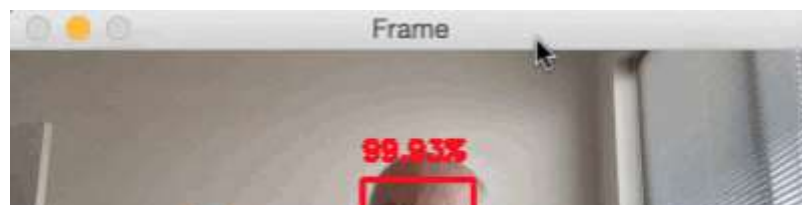
- The **source code** used in this blog post
- The **Caffe prototxt files** for deep learning face detection
- The **Caffe weight files** used for deep learning face detection

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

→ [Launch Jupyter Notebook on Google Colab](#)

Face detection with OpenCV and deep learning

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



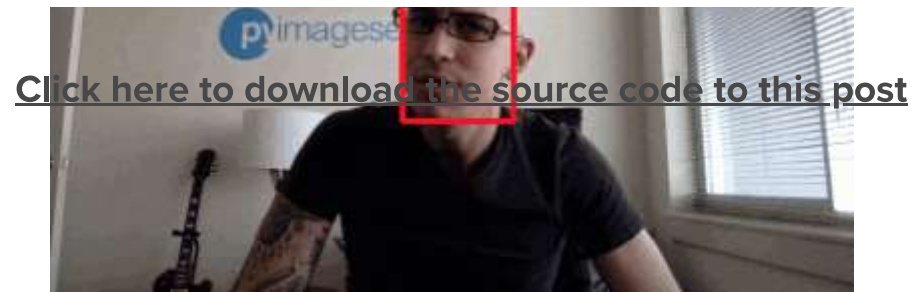
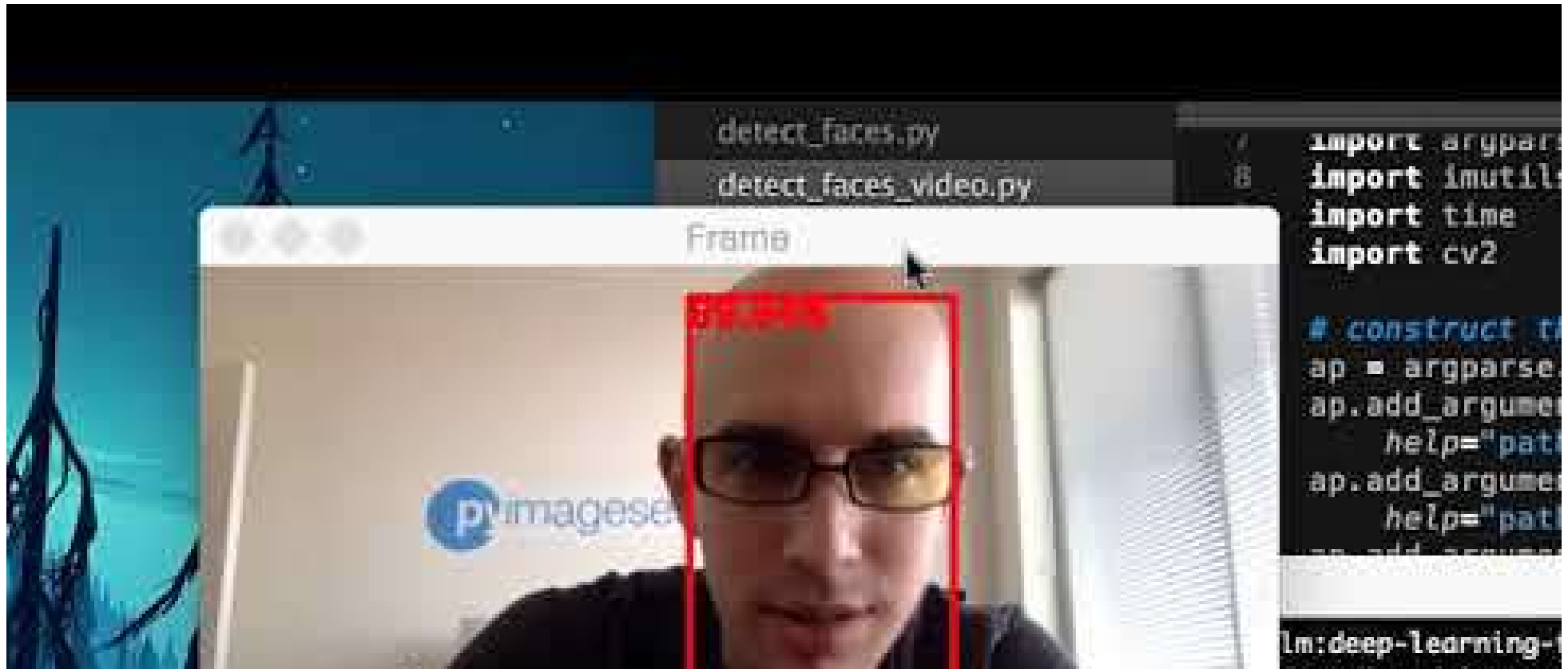
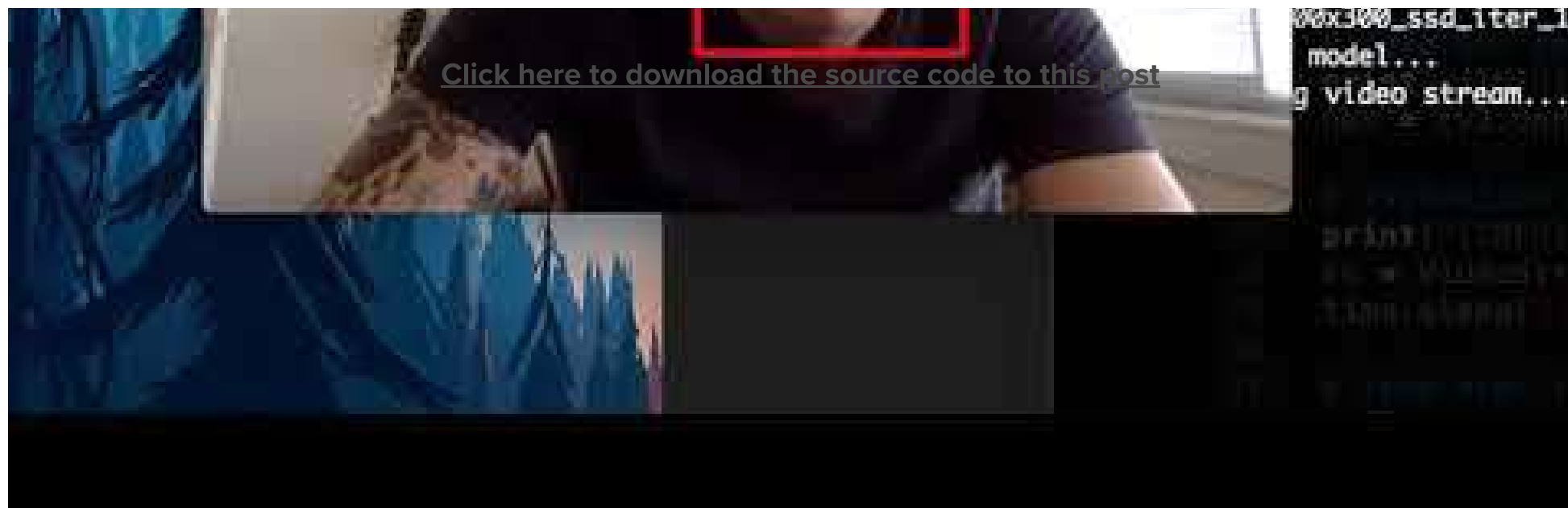


Figure 5: Face detection in video with OpenCV's DNN module.

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





Other face detection methods to consider

This tutorial covered how to use OpenCV's "hidden" pre-trained deep learning face detector. This model is a good balance of both *speed* and *accuracy*.

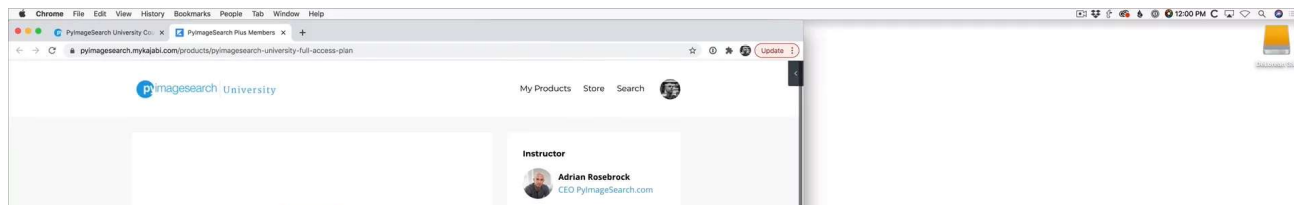
However, there are other face detection methods that you may want to consider for your projects:

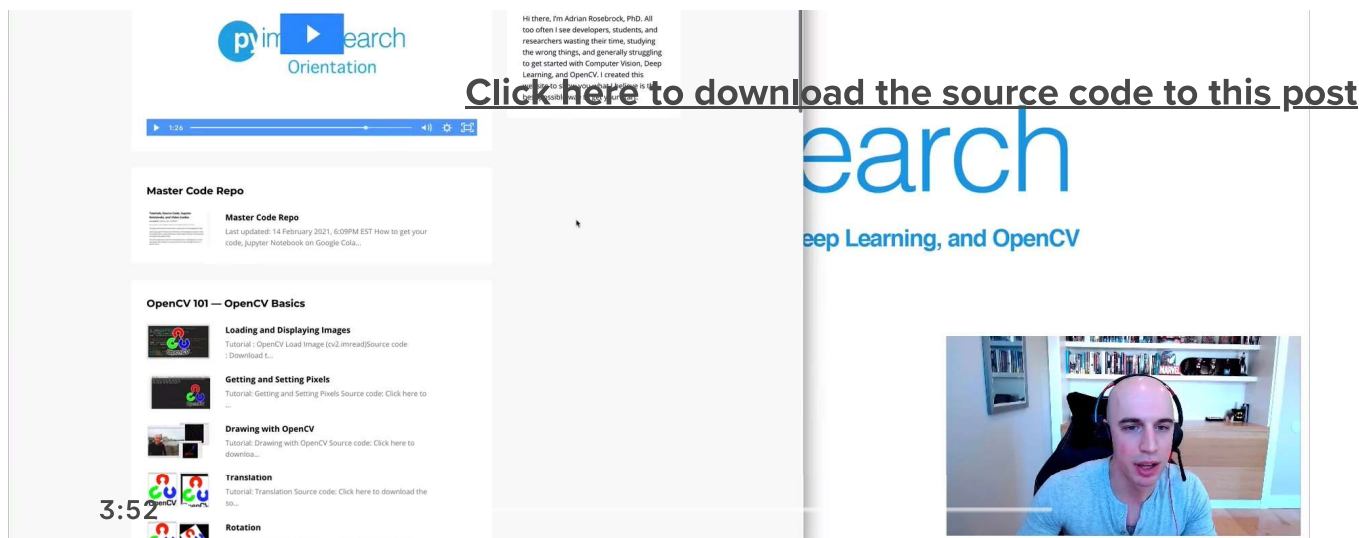
- 1 **Face detection with Haar cascades (<https://www.pyimagesearch.com/2021/04/05/opencv-face-detection-with-haar-cascades/>):** *Extremely fast* but prone to false-positives and in general less accurate than deep learning-based face detectors

- 2 **Face detection with dlib (HOG and CNN) (<https://www.pyimagesearch.com/2021/04/19/face-detection-with-dlib-hog-and-cnn/>):** ~~HOG is more accurate than Haar cascades but computationally more expensive.~~ **Click here to download the source code to this post** Dlib's CNN face detector is the most accurate of the bunch but cannot run in real-time without a GPU.
- 3 **Multi-task Cascaded Convolutional Networks (MTCNNs) (<https://arxiv.org/abs/1604.02878>):** Very accurate deep learning-based face detector. **Easily compatible with both Keras and TensorFlow (<https://github.com/ipazc/mtcnn>).**

Finally, I *highly suggest* you read my **[Face detection tips, suggestions, and best practices](https://www.pyimagesearch.com/2021/04/26/face-detection-tips-suggestions-and-best-practices/)** (<https://www.pyimagesearch.com/2021/04/26/face-detection-tips-suggestions-and-best-practices/>) tutorial where I detail the pros and cons of each face detection method.

What's next? I recommend [PyImageSearch University](https://www.pyimagesearch.com/pyimagesearch-university/) (https://www.pyimagesearch.com/pyimagesearch-university/?utm_source=blogPost&utm_medium=bottomBanner&utm_campaign=What%27s%20next%3F%20I%20recommend).





Click here to download the source code to this post

Course information:

28 total classes • 39h 44m video • Last updated: 10/2021

★★★★★ 4.84 (128 Ratings) • 3,000+ Students Enrolled

I strongly believe that if you had the right teacher you could *master* computer vision and deep learning.

Do you think learning computer vision and deep learning has to be time-consuming, overwhelming, and complicated? Or has to involve complex mathematics and equations? Or requires a degree in computer science?

That's *not* the case.

[Click here to download the source code to this post](#)

All you need to master computer vision and deep learning is for someone to explain things to you in *simple, intuitive* terms. *And that's exactly what I do.* My mission is to change education and how complex Artificial Intelligence topics are taught.

If you're serious about learning computer vision, your next stop should be PyImageSearch University, the most comprehensive computer vision, deep learning, and OpenCV course online today. Here you'll learn how to *successfully* and *confidently* apply computer vision to your work, research, and projects. Join me in computer vision mastery.

Inside PyImageSearch University you'll find:

- ✓ **28 courses** on essential computer vision, deep learning, and OpenCV topics
- ✓ 28 Certificates of Completion
- ✓ **39h 44m** on-demand video
- ✓ **Brand new courses released every month**, ensuring you can keep up with state-of-the-art techniques
- ✓ **Pre-configured Jupyter Notebooks in Google Colab**
- ✓ **Run all code examples in your web browser** — works on Windows, macOS, and Linux (no dev

✓ Run all code examples in your web browser — works on windows, macOS, and Linux (no dev environment configuration required!)

[Click here to download the source code to this post](#)

✓ Access to **centralized code repos for all 400+ tutorials** on PyImageSearch

✓ **Easy one-click downloads** for code, datasets, pre-trained models, etc.

✓ Access on mobile, laptop, desktop, etc.

CLICK HERE TO JOIN PYIMAGESEARCH UNIVERSITY

([HTTPS://WWW.PYIMAGESEARCH.COM/PYIMAGESEARCH-UNIVERSITY/?](https://www.pyimagesearch.com/pyimagesearch-university/?)

UTM_SOURCE=BLOGPOST&UTM_MEDIUM=BOTTOMBANNER&UTM_CAMPAIGN=WHAT%27S%20NEXT%3F%20I%20RECOMMEND)

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.

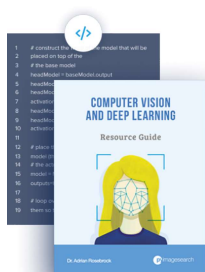
Thanks to the hard work of Aleksandr Rybnikov and the other contributors to OpenCV's `dnn` module, we can enjoy

these more accurate OpenCV face detectors in our own applications.

[Click here to download the source code to this post](#)

The deep learning face detectors can be hard to find in the OpenCV library, so 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!



Download the Source Code and FREE 17-page Resource Guide

Enter your email address below to get a .zip of the code and a **FREE 17-page Resource Guide on Computer Vision, OpenCV, and Deep Learning.** Inside you'll find my hand-picked tutorials, books, courses, and libraries to help you master CV and DL!



About the Author



ABOUT THE AUTHOR

Hi there, I'm Adrian Rosebrock, PhD. All too often I see developers, students, and researchers wasting their time, studying the wrong things, and generally struggling to get started with Computer Vision, Deep Learning, and OpenCV. I created this website to show you what I believe is the best possible way to get your start.

[Click here to download the source code to this post](#)

Previous Article:

Real-time object detection on the Raspberry Pi with the Movidius NCS

(<https://www.pyimagesearch.com/2018/02/19/real-time-object-detection-on-the-raspberry-pi-with-the-movidius-ncs/>)

Next Article:

Python, argparse, and command line arguments

(<https://www.pyimagesearch.com/2018/03/12/python-argparse-command-line-arguments/>)

456 responses to: Face detection with OpenCV and deep learning