Click here to download the source code to this post

PYIMAGESEARCH

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

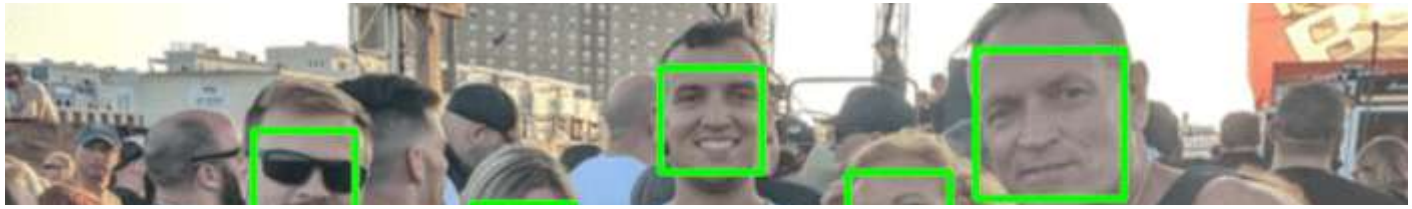
FACE APPLICATIONS (HTTPS://WWW.PYIMAGESEARCH.COM/CATEGORY/FACES/)

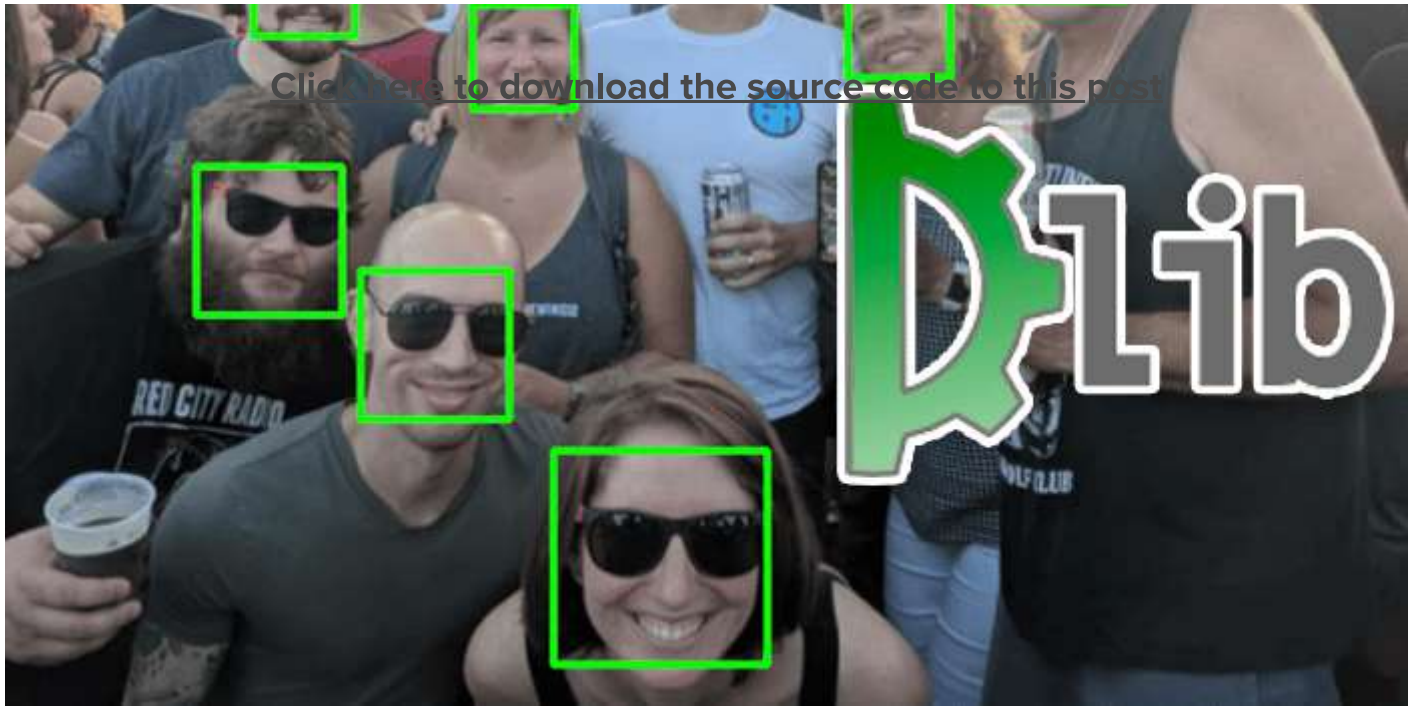
TUTORIALS (HTTPS://WWW.PYIMAGESEARCH.COM/CATEGORY/TUTORIALS/)

Face detection with dlib (HOG and CNN)

by **Adrian Rosebrock (https://www.pyimagesearch.com/author/adrian/)** *on* April 19, 2021

In this tutorial, you will learn how to perform face detection with the dlib library using both HOG + Linear SVM and CNNs.





The dlib library is arguably one of *the most utilized packages* for face recognition. A Python package appropriately named `face_recognition` wraps dlib's face recognition functions into a simple, easy to use API.

Note: *If you are interested in using dlib and the* `face_recognition` *libraries for face recognition,* ***refer to this tutorial*** **(https://www.pyimagesearch.com/2018/06/18/face-recognition-with-opencv-python-and-deep-learning/)**, *where I cover the topic in detail.*

However, I'm often surprised to hear that readers do not know that dlib includes two face detection methods built into the library:

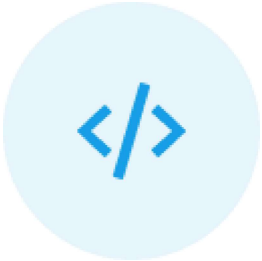
1. A **HOG + Linear SVM face detector** that is accurate and computationally efficient.

2. A **Max-Margin (MMOD) CNN face detector** that is both *highly accurate* and very *robust*, capable of detecting faces from varying viewing angles, lighting conditions, and occlusion.

Click here to download the source code to this post

Best of all, the MMOD face detector can run on an NVIDIA GPU, making it super fast!

To learn how to use dlib's HOG + Linear SVM and MMOD face detectors, *just keep reading*.



Looking for the source code to this post?

JUMP RIGHT TO THE DOWNLOADS SECTION →

Face detection with dlib (HOG and CNN)

In the first part of this tutorial, you'll discover dlib's two face detection functions, one for a HOG + Linear SVM face

detector and another for the MMOD CNN face detector.

Click here to download the source code to this post

From there, we'll configure our development environment and review our project directory structure.

We'll then implement two Python scripts:

1. `hog_face_detection.py` : Applies dlib's HOG + Linear SVM face detector.
2. `cnn_face_detection.py` : Utilizes dlib's MMOD CNN face detector.

We'll then run these face detectors on a set of images and examine the results, noting when to use each face detector in a given situation.

Let's get started!

Dlib's face detection methods

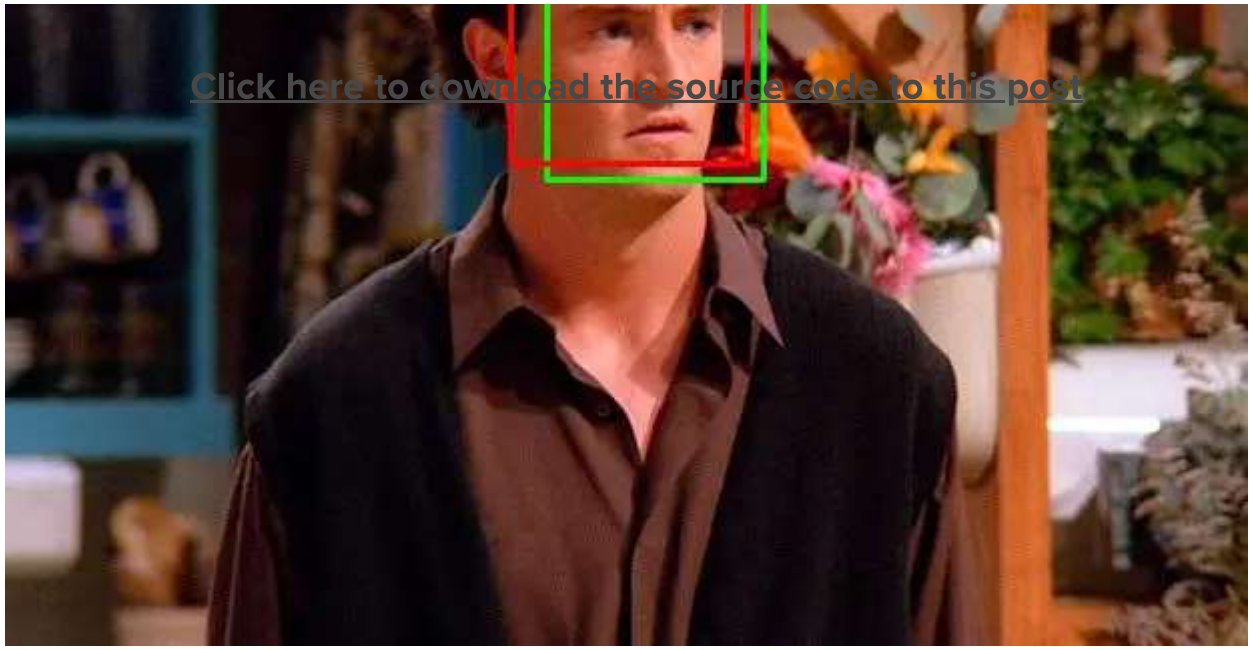


Figure 1: The dlib library provides two functions for face detection. The first one is a HOG + Linear SVM face detector, and the other is a deep learning MMOD CNN face detector (**image source (https://towardsdatascience.com/cnn-based-face-detector-from-dlib-c3696195e01c)**).

The dlib library provides two functions that can be used for face detection:

1. **HOG + Linear SVM:** `dlib.get_frontal_face_detector()`
2. **MMOD CNN:** `dlib.cnn_face_detection_model_v1(modelPath)`

The `get_frontal_face_detector` function does not accept any parameters. A call to it returns the pre-trained HOG + Linear SVM face detector included in the dlib library.

Dlib's HOG + Linear SVM face detector is fast and efficient. By nature of how the Histogram of Oriented Gradients

(HOG) descriptor works, it is *not* invariant to changes in rotation and viewing angle.

Click here to download the source code to this post

For more robust face detection, you can use the MMOD CNN face detector, available via the `cnn_face_detection_model_v1` function. This method accepts a single parameter, `modelPath`, which is the path to the pre-trained `mmod_human_face_detector.dat` file residing on disk.

Note: *I've included the* `mmod_human_face_detector.dat` *file in the* ***"Downloads"*** *section of this guide, so you don't have to go hunting for it.*

In the remainder of this tutorial, you will learn how to use both of these dlib face detection methods.

Configuring your development environment

To follow this guide, you need to have both the OpenCV library and dlib installed on your system.

Luckily, you can install OpenCV and dlib via pip:

→ **Launch Jupyter Notebook on Google Colab**

```
Face detection with dlib (HOG and CNN)
$ pip install opencv-contrib-python
$ pip install dlib
```

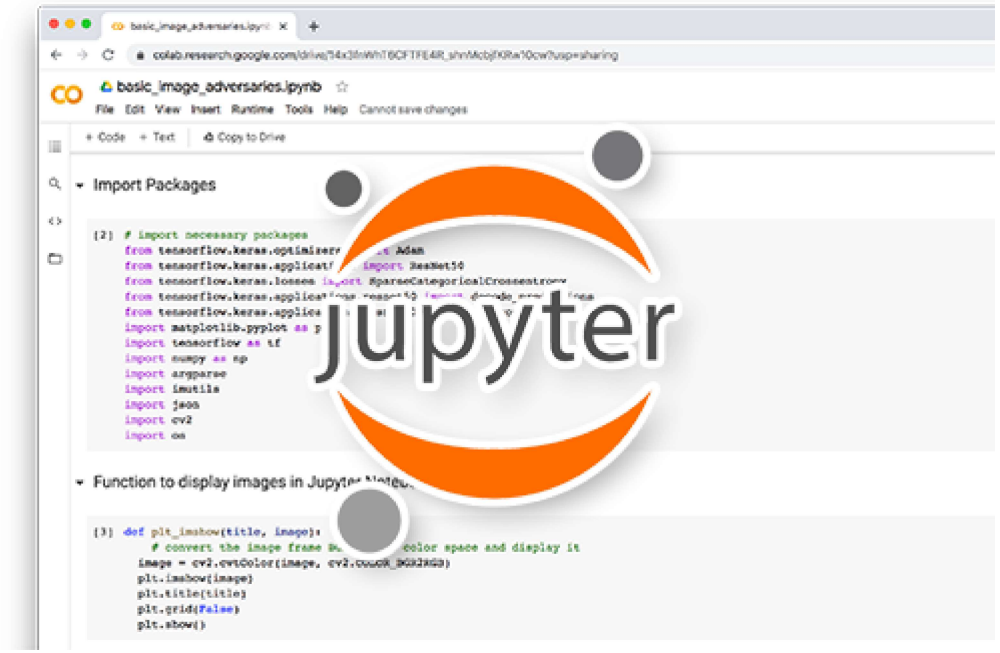
If you need help configuring your development environment for OpenCV and dlib, I *highly recommend* that you read the following two tutorials:

1. **pip install opencv (https://www.pyimagesearch.com/2018/09/19/pip-install-opencv/)**

2 How to install dlib (https://www.pyimagesearch.com/2018/01/22/install-dlib-easy-complete-guide/)

Click here to download the source code to this post

Having problems configuring your development environment?



(https://www.pyimagesearch.com/pyimagesearch-university/)

Figure 2: Having trouble configuring your dev environment? Want access to pre-configured Jupyter Notebooks running on Google Colab? Be sure to join

PyImageSearch University (https://www.pyimagesearch.com/pyimagesearch-university/) — you'll be up and running with this tutorial in a matter of minutes.

All that said, are you:

- Short on time?

Click here to download the source code to this post

- Learning on your employer's administratively locked system?
- Wanting to skip the hassle of fighting with the command line, package managers, and virtual environments?
- **Ready to run the code *right now* on your Windows, macOS, or Linux systems?**

Then join **PyImageSearch University (https://www.pyimagesearch.com/pyimagesearch-university/)** today!

Gain access to Jupyter Notebooks for this tutorial and other PyImageSearch guides that are *pre-configured* to run on Google Colab's ecosystem right in your web browser! No installation required.

And best of all, these Jupyter Notebooks will run on Windows, macOS, and Linux!

Project structure

Before we can perform face detection with dlib, we first need to review our project directory structure.

Start by accessing the ***"Downloads"*** section of this tutorial to retrieve the source code and example images.

From there, take a look at the directory structure:

→ **Launch Jupyter Notebook on Google Colab**

```
Face detection with dlib (HOG and CNN)
$ tree . --dirsfirst
.
```

```
3. |   ├── images
4. |   |   ├── avengers.jpg
5. |   |   ├── concert.jpg
6. |   |   └── family.jpg
7. |   ├── pyimagesearch
8. |   |   ├── __init__.py
9. |   |   └── helpers.py
10. |  ├── cnn_face_detection.py
11. |  ├── hog_face_detection.py
12. |  └── mmod_human_face_detector.dat
```

Click here to download the source code to this post

We start with two Python scripts to review:

1. `hog_face_detection.py` : Applies HOG + Linear SVM face detection using dlib.
2. `cnn_face_detection.py` : Performs deep learning-based face detection using dlib by loading the trained `mmod_human_face_detector.dat` model from disk.

Our `helpers.py` file contains a Python function, `convert_and_trim_bb` , which will help us:

1. Convert dlib bounding boxes to OpenCV bounding boxes
2. Trim any bounding box coordinates that fall outside the bounds of the input image

The `images` directory contains three images that we'll be applying face detection to with dlib. We can compare the HOG + Linear SVM face detection method with the MMOD CNN face detector.

Creating our dlib face detection and dlib to OpenCV functions

Creating our bounding box converting and clipping function

Click here to download the source code to this post

OpenCV and dlib represent bounding boxes differently:

- In OpenCV, we think of bounding boxes in terms of a 4-tuple of starting *x*-coordinate, starting *y*-coordinate, width, and height
- Dlib represents bounding boxes via `rectangle` object with left, top, right, and bottom properties

Furthermore, bounding boxes returned by dlib may fall *outside* the bounds of the input image dimensions (negative values or values outside the width and height of the image).

To make applying face detection with dlib easier, let's create a helper function to (1) convert the bounding box coordinates to standard OpenCV ordering and (2) trim any bounding box coordinates that fall outside the image's range.

Open the `helpers.py` file inside the `pyimagesearch` module, and let's get to work:

→ **Launch Jupyter Notebook on Google Colab**

Face detection with dlib (HOG and CNN)

```
def convert_and_trim_bb(image, rect):
    # extract the starting and ending (x, y)-coordinates of the
    # bounding box

    startX = rect.left()
    startY = rect.top()
    endX = rect.right()
    endY = rect.bottom()

    # ensure the bounding box coordinates fall within the spatial
    # dimensions of the image
```

```
10.     # dimensions of the image
11.     startX = max(0, startX)
12.     startY = max(0, startY)
13.     endX = min(endX, image.shape[1])
14.     endY = min(endY, image.shape[0])
15.
16.     # compute the width and height of the bounding box
17.     w = endX - startX
18.     h = endY - startY
19.
20.     # return our bounding box coordinates
21.     return (startX, startY, w, h)
```


Our `convert_and_trim_bb` function requires two parameters: the input `image` we applied face detection to and the `rect` object returned by dlib.

Lines 4-7 extract the starting and ending *(x, y)*-coordinates of the bounding box.

We then ensure the bounding box coordinates fall within the width and height of the input `image` on **Lines 11-14.**

The final step is to compute the width and height of the bounding box (**Lines 17 and 18**) and then return a 4-tuple of the bounding box coordinates in `startX`, `startY`, `w`, and `h` order.

Implementing HOG + Linear SVM face detection with dlib

With our `convert_and_trim_bb` helper utility implemented, we can move on to perform HOG + Linear SVM face detection using dlib.

Open the `hog_face_detection.py` file in your project directory structure and insert the following code:

→ Launch Jupyter Notebook on Google Colab **Click here to download the source code to this post**

```
Face detection with dlib (HOG and CNN)
1. | # import the necessary packages
2. | from pyimagesearch.helpers import convert_and_trim_bb
3. | import argparse
4. | import imutils
5. | import time
6. | import dlib
7. | import cv2
```

Lines 2-7 import our required Python packages. Notice that the `convert_and_trim_bb` function we just implemented is imported.

While we import `cv2` for our OpenCV bindings, we also import `dlib`, so we can access its face detection functionality.

Next is our command line arguments:

→ Launch Jupyter Notebook on Google Colab

```
Face detection with dlib (HOG and CNN)
9. | # construct the argument parser and parse the arguments
10. | ap = argparse.ArgumentParser()
11. | ap.add_argument("-i", "--image", type=str, required=True,
12. |    help="path to input image")
13. | ap.add_argument("-u", "--upsample", type=int, default=1,
14. |    help="# of times to upsample")
15. | args = vars(ap.parse_args())
```

We have two command line arguments to parse:

1. `--image` : The path to the input image where we apply HOG + Linear SVM face detection.

Click here to download the source code to this post

2. `--upsample` : Number of times to upsample an image before applying face detection.

To detect small faces in a large input image, we may wish to increase the resolution of the input image, thereby making the smaller faces appear larger. Doing so allows our **sliding window (https://www.pyimagesearch.com/2015/03/23/sliding-windows-for-object-detection-with-python-and-opencv/)** to detect the face.

The downside to upsampling is that it creates more layers of our **image pyramid (https://www.pyimagesearch.com/2015/03/16/image-pyramids-with-python-and-opencv/)**, making the detection process slower.

For faster face detection, set the `--upsample` value to `0`, meaning that *no* upsampling is performed (but you risk missing face detections).

Next, let's load dlib's HOG + Linear SVM face detector from disk:

→ **Launch Jupyter Notebook on Google Colab**

```
Face detection with dlib (HOG and CNN)
# load dlib's HOG + Linear SVM face detector
print("[INFO] loading HOG + Linear SVM face detector...")
detector = dlib.get_frontal_face_detector()

# load the input image from disk, resize it, and convert it from
# BGR to RGB channel ordering (which is what dlib expects)
image = cv2.imread(args["image"])
image = imutils.resize(image, width=600)
```

```
25. | rgb = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
26. |
27. | # perform face detection using dlib's face detector
28. | start = time.time()
29. | print("[INFO[ performing face detection with dlib...")
30. | rects = detector(rgb, args["upsample"])
31. | end = time.time()
32. | print("[INFO] face detection took {:.4f} seconds".format(end - start))
```


A call to `dlib.get_frontal_face_detector()` returns dlib's HOG + Linear SVM face detector (**Line 19**).

We then proceed to:

1. Load the input `image` from disk
2. Resize the image (the smaller the image is, the faster HOG + Linear SVM will run)
3. Convert the image from BGR to RGB channel ordering (dlib expects RGB images)

From there, we apply our HOG + Linear SVM face detector on **Line 30**, timing how long the face detection process takes.

Let's now parse our bounding boxes:


```
Face detection with dlib (HOG and CNN)
34. | # convert the resulting dlib rectangle objects to bounding boxes,
35. | # then ensure the bounding boxes are all within the bounds of the
36. | # input image
37. | boxes = [convert_and_trim_bb(image, r) for r in rects]
38. |
```

```
39. | # loop over the bounding boxes
40. | for (x, y, w, h) in boxes:
41. |    # draw the bounding box on our image
42. |    cv2.rectangle(image, (x, y), (x + w, y + h), (0, 255, 0), 2)
43. |
44. | # show the output image
45. | cv2.imshow("Output", image)
46. | cv2.waitKey(0)
```

Click here to download the source code to this post

Keep in mind that the returned `rects` list needs some work — we need to parse the dlib `rectangle` objects into a 4-tuple of starting *x*-coordinate, starting *y*-coordinate, width, and height — and that's *exactly* what **Line 37** accomplishes.

For each `rect`, we call our `convert_and_trim_bb` function, ensuring that both (1) all bounding box coordinates fall within the spatial dimensions of the `image` and (2) our returned bounding boxes are in the proper 4-tuple format.

Dlib HOG + Linear SVM face detection results

Let's look at the results of applying our dlib HOG + Linear SVM face detector to a set of images.

Be sure to access the ***"Downloads"*** section of this tutorial to retrieve the source code, example images, and pre-trained models.

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

→ **Launch Jupyter Notebook on Google Colab**

```
Face detection with dlib (HOG and CNN)
1. | $ python hog_face_detection.py --image images/family.jpg
2. | [INFO] loading HOG + Linear SVM face detector...
```

```
3. | [INFO[ performing face detection with dlib...
4. | [INFO] face detection took 0.1062 seconds
```

Click here to download the source code to this post

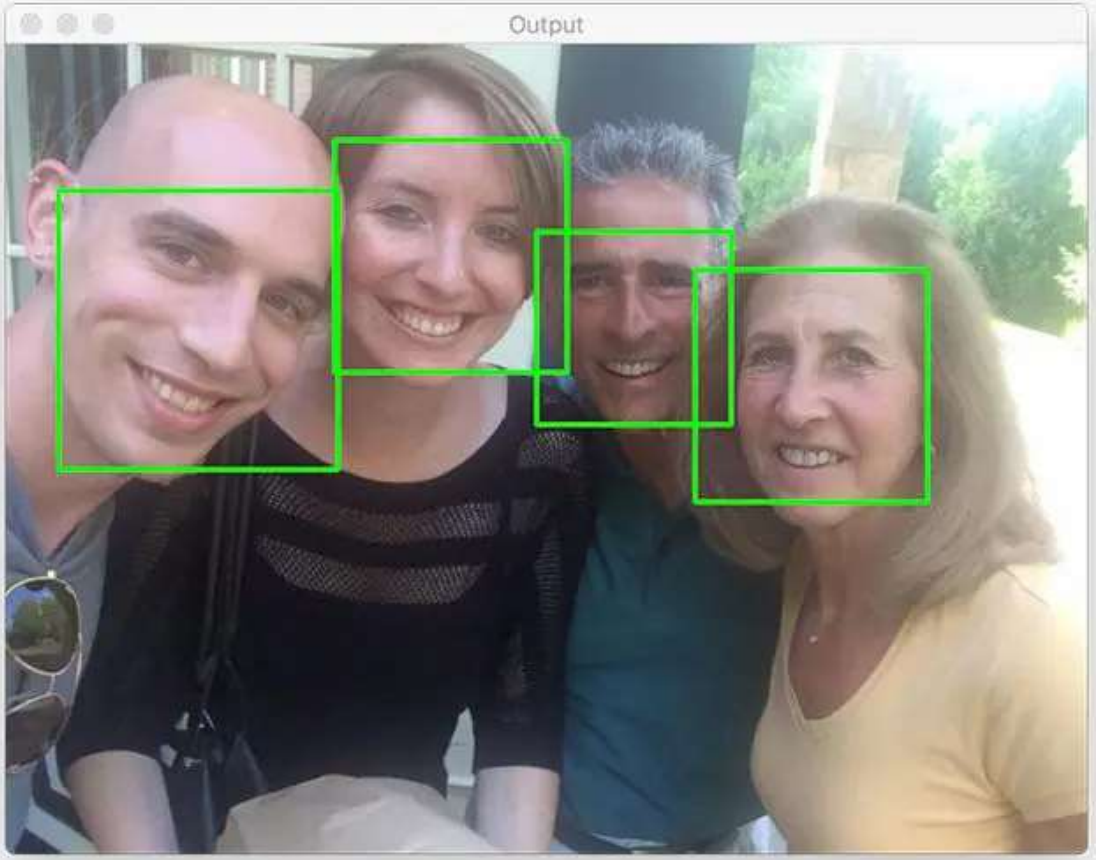


Figure 3: Successfully applying dlib's HOG + Linear SVM face detector.

Figure 3 displays the results of applying dlib's HOG + Linear SVM face detector to an input image containing multiple f

faces.

Click here to download the source code to this post

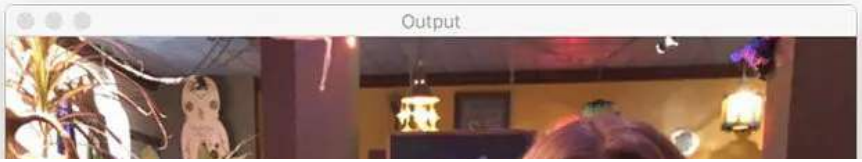
The face detection process took ~0.1 seconds, implying that we could process ~10 frames per second in a video stream scenario.

Most importantly, note that each of the four faces was correctly detected.

Let's try a different image:

→ **Launch Jupyter Notebook on Google Colab**

```
Face detection with dlib (HOG and CNN)
$ python hog_face_detection.py --image images/avengers.jpg
[INFO] loading HOG + Linear SVM face detector...
[INFO[ performing face detection with dlib...
[INFO] face detection took 0.1425 seconds
```



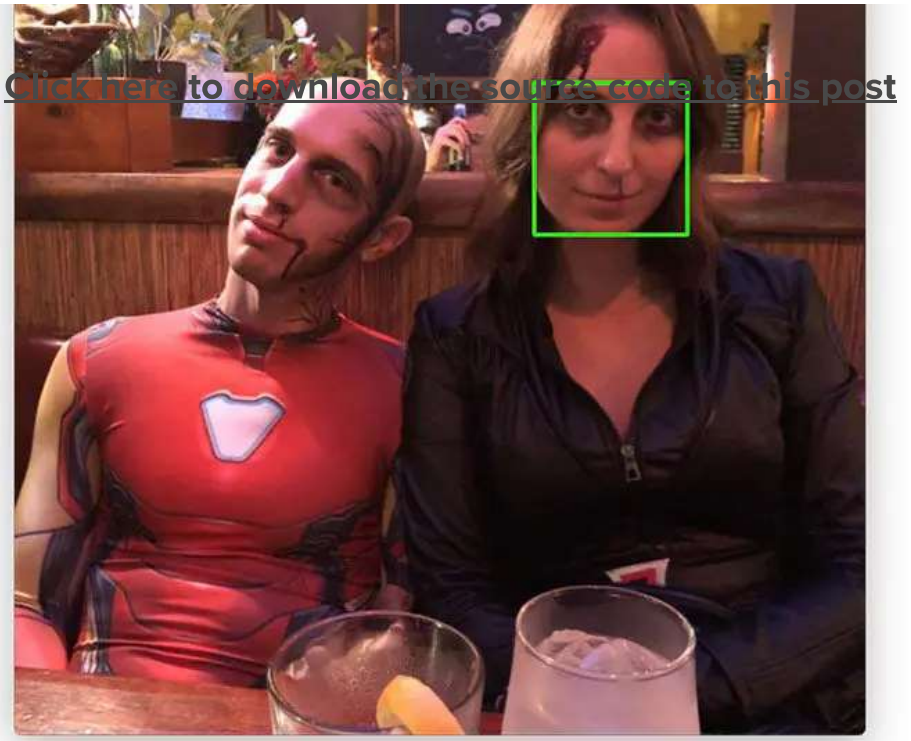


Figure 4: Dlib's HOG + Linear SVM face detector fails to detect a face.

A couple of years ago, back when *Avengers: Endgame* came out, my wife and I decided to dress up as "dead Avengers" from the movie (sorry if you haven't seen the movie but come on, it's been two years already!)

Notice that my wife's face (errr, Black Widow?) was detected, but apparently, dlib's HOG + Linear SVM face detector doesn't know what Iron Man looks like.

In all likelihood, my face wasn't detected because my head is slightly rotated and is not a "straight-on view" for the

camera. Again, the HOG + Linear SVM family of object detectors *does not* perform well under rotation or viewing angle changes.

Click here to download the source code to this post

Let's look at one final image, this one more densely packed with faces:

→ **Launch Jupyter Notebook on Google Colab**

```
Face detection with dlib (HOG and CNN)
$ python hog_face_detection.py --image images/concert.jpg
[INFO] loading HOG + Linear SVM face detector...
[INFO[ performing face detection with dlib...
[INFO] face detection took 0.1069 seconds
```

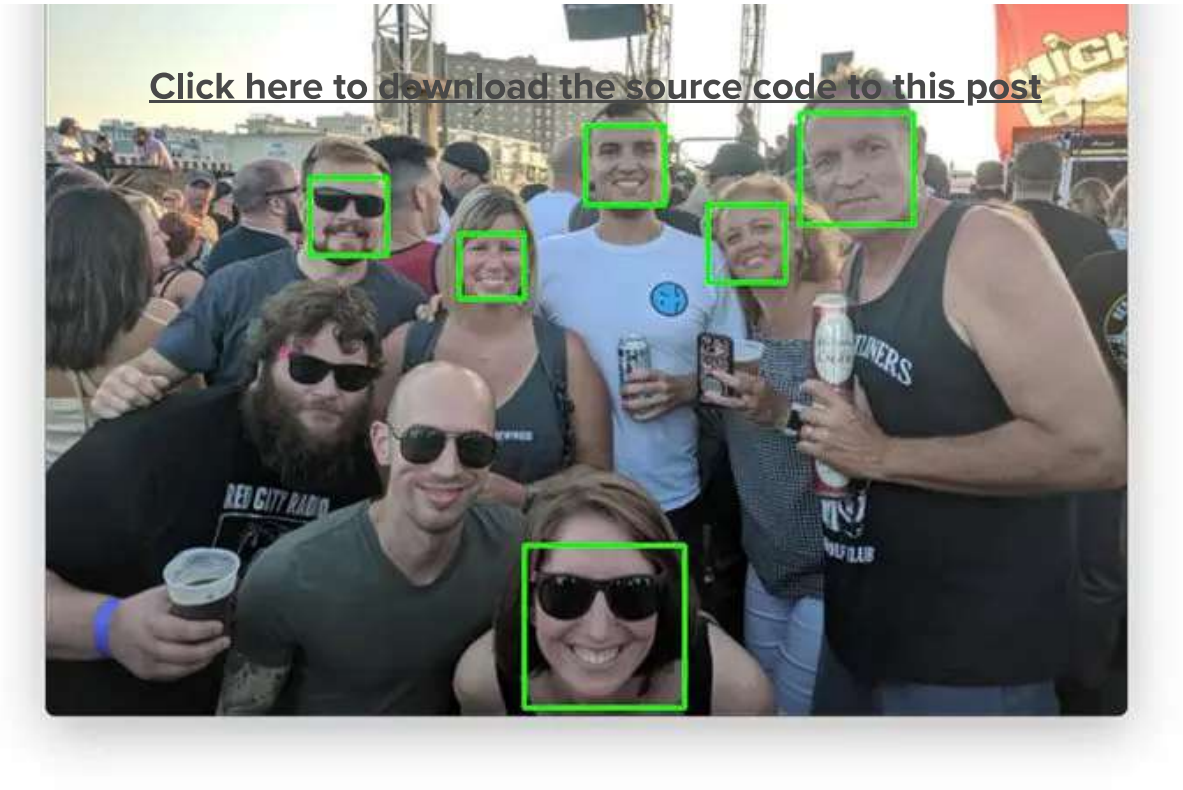


Figure 5: Applying dlib's HOG + Linear SVM face detector to many faces.

Back before COVID, there were these things called "concerts." Bands used to get together and play live music for people in exchange for money. Hard to believe, I know.

A bunch of my friends got together for a concert a few years ago. And while there are clearly eight faces in this image, only six of them are detected.

As we'll see later in this tutorial, we can use dlib's MMOD CNN face detector to improve face detection accuracy and

detect *all* the faces in this image.

Click here to download the source code to this post

Implementing CNN face detection with dlib

So far, we have learned how to perform face detection with dlib's HOG + Linear SVM model. This method worked well, but there is *far* more accuracy to be obtained by using dlib's MMOD CNN face detector.

Let's learn how to use dlib's deep learning face detector now:

→ **Launch Jupyter Notebook on Google Colab**

Face detection with dlib (HOG and CNN)

```
1. # import the necessary packages
2. from pyimagesearch.helpers import convert_and_trim_bb
3. import argparse
4. import imutils
5. import time
6. import dlib
7. import cv2
```

Our imports here are *identical* to our previous script on HOG + Linear SVM face detection.

The command line arguments are similar, but with one addition (the `--model`) argument:

→ **Launch Jupyter Notebook on Google Colab**

Face detection with dlib (HOG and CNN)

```
9. # construct the argument parser and parse the arguments
10. ap = argparse.ArgumentParser()
11. ap.add_argument("-i", "--image", type=str, required=True,
12.    help="path to input image")
13. ap.add_argument("-m", "--model", type=str
```

```
    ap.add_argument("-m", "--model", type=str,
        default="mmod_human_face_detector.dat",
        help="path to dlib's CNN face detector model")
    ap.add_argument("-u", "--upsample", type=int, default=1,
        help="# of times to upsample")
    args = vars(ap.parse_args())
```

Click here to download the source code to this post

We have three command line arguments here:

1. `--image` : The path to the input image residing on disk.
2. `--model` : Our pre-trained dlib MMOD CNN face detector.
3. `--upsample` : The number of times to upsample an image before applying face detection.

With our command line arguments taken care of, we can now load dlib's deep learning face detector from disk:

→ **Launch Jupyter Notebook on Google Colab**

```
Face detection with dlib (HOG and CNN)
# load dlib's CNN face detector
print("[INFO] loading CNN face detector...")
detector = dlib.cnn_face_detection_model_v1(args["model"])

# load the input image from disk, resize it, and convert it from
# BGR to RGB channel ordering (which is what dlib expects)
image = cv2.imread(args["image"])
image = imutils.resize(image, width=600)

rgb = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)

# perform face detection using dlib's face detector
start = time.time()
print("[INFO[ performing face detection with dlib...")
results = detector(rgb, args["upsample"])
end = time.time()
```

```
34. | end = time.time()
35. | print("[INFO] face detection took {:.4f} seconds".format(end - start))
```

Click here to download the source code to this post

Line 22 loads the `detector` from disk by calling `dlib.cnn_face_detection_model_v1`. Here we pass in `--model`, the path to where the trained dlib face detector resides.

From there, we preprocess our image (**Lines 26-28**) and then apply the face detector (**Line 33**).

Just as we parsed the HOG + Linear SVM results, we need to the same here, but one with one caveat:

→ **Launch Jupyter Notebook on Google Colab**

Face detection with dlib (HOG and CNN)

```
37. | # convert the resulting dlib rectangle objects to bounding boxes,
38. | # then ensure the bounding boxes are all within the bounds of the
39. | # input image
40. | boxes = [convert_and_trim_bb(image, r.rect) for r in results]
41. |
42. | # loop over the bounding boxes
43. | for (x, y, w, h) in boxes:
44. |     # draw the bounding box on our image
45. |     cv2.rectangle(image, (x, y), (x + w, y + h), (0, 255, 0), 2)
46. |
47. | # show the output image
48. | cv2.imshow("Output", image)
49. | cv2.waitKey(0)
```

Dlib's HOG + Linear SVM detector returns a list of `rectangle` objects; however, the MMOD CNN object detector returns a list of result objects, each with its own rectangle (hence we use `r.rect` in the list comprehension). Otherwise, the implementation is the same.

Finally, we loop over the bounding boxes and draw them on our output `image`.

Click here to download the source code to this post

Dlib's CNN face detector results

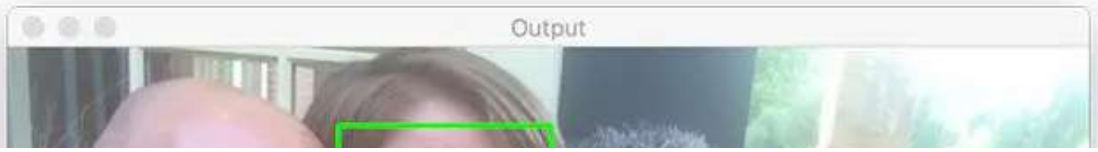
Let's see how dlib's MMOD CNN face detector stacks up to the HOG + Linear SVM face detector.

To follow along, be sure to access the ***"Downloads"*** section of this guide to retrieve the source code, example images, and pre-trained dlib face detector.

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

→ **Launch Jupyter Notebook on Google Colab**

```
Face detection with dlib (HOG and CNN)
$ python cnn_face_detection.py --image images/family.jpg
[INFO] loading CNN face detector...
[INFO[ performing face detection with dlib...
[INFO] face detection took 2.3075 seconds
```



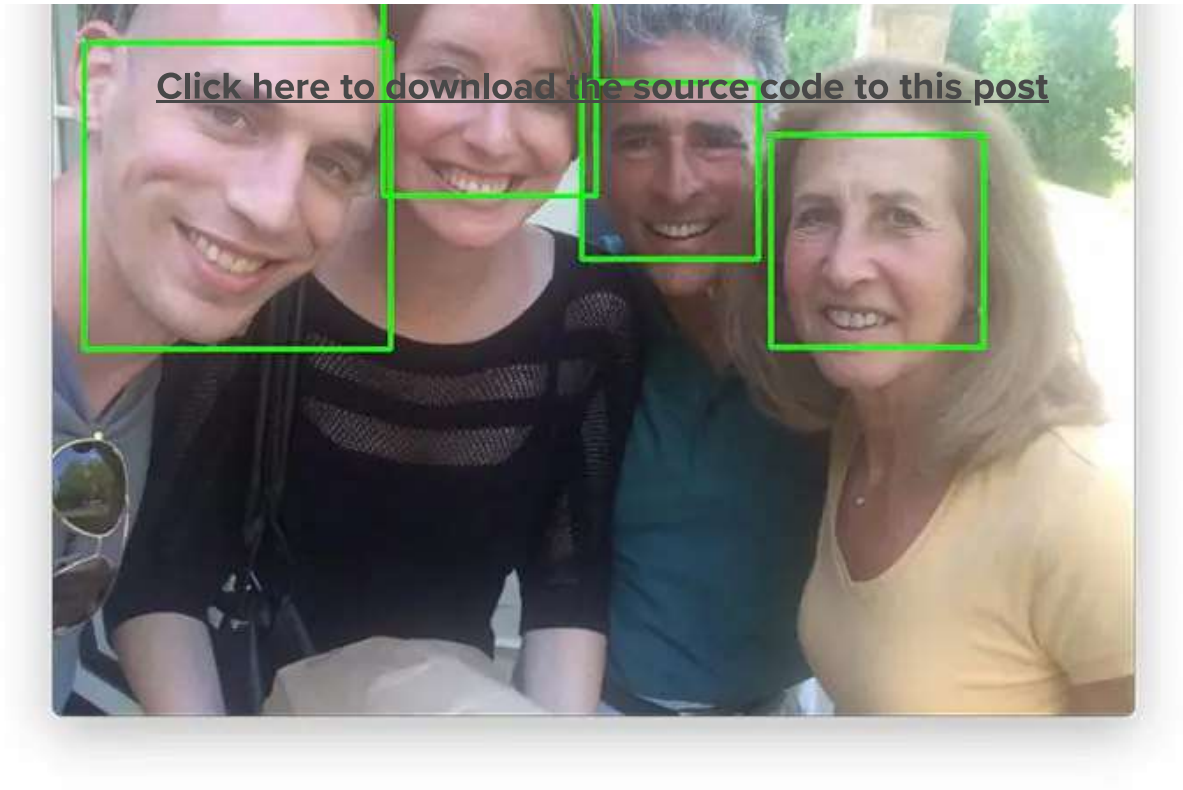


Figure 6: Using dlib's deep learning MMOD CNN face detector.

Just like the HOG + Linear SVM implementation, dlib's MMOD CNN face detector can correctly detect all four faces in the input image.

Let's try another image:

→ **Launch Jupyter Notebook on Google Colab**

```
Face detection with dlib (HOG and CNN)
$ python cnn_face_detection.py --image images/avengers.jpg
[INFO] loading CNN face detector...
```

```
3. | [INFO[ performing face detection with dlib...
4. | [INFO] face detection took 3.0468 seconds
```

Click here to download the source code to this post

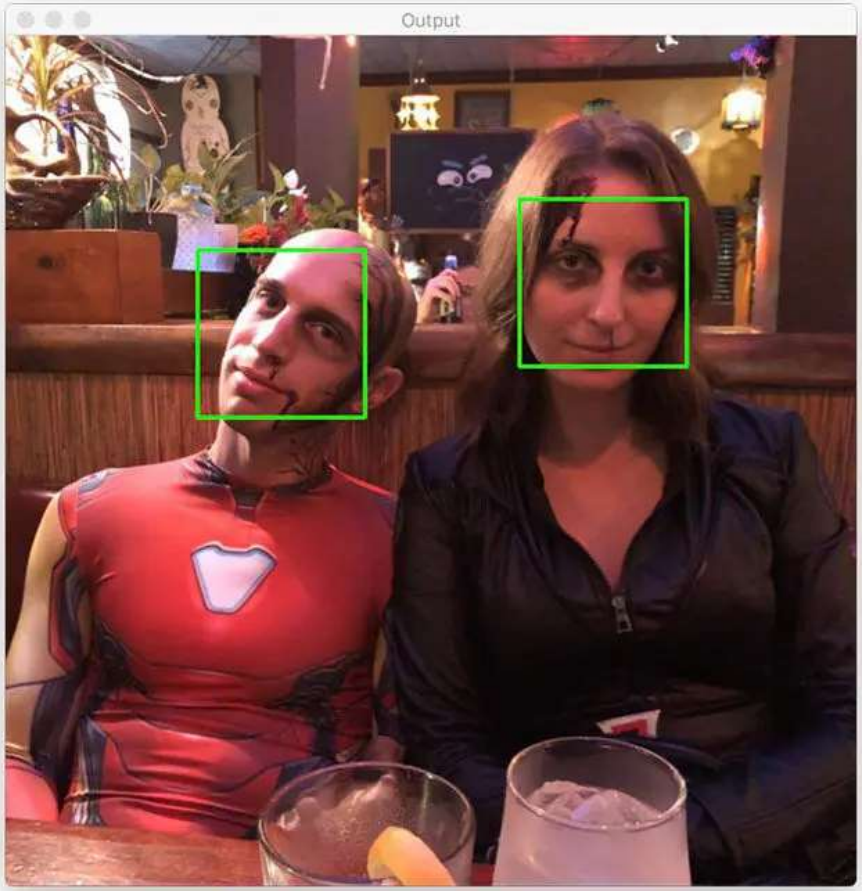


Figure 7: Dlib's deep learning-based face detector can detect the face that the HOG + Linear SVM method missed.

Previously, HOG + Linear SVM failed to detect my face on the left. But by using dlib's deep learning face detector, we

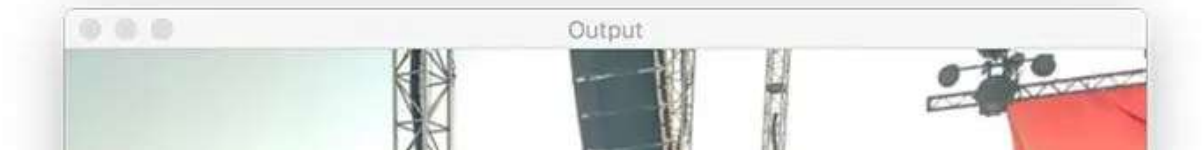
Previously, HOG + Linear SVM failed to detect my face on the left. But by using dlib's deep learning face detector, we can correctly detect *both* faces.

Click here to download the source code to this post

Let's look at one final image:

→ **Launch Jupyter Notebook on Google Colab**

```
Face detection with dlib (HOG and CNN)
$ python cnn_face_detection.py --image images/concert.jpg
[INFO] loading CNN face detector...
[INFO[ performing face detection with dlib...
[INFO] face detection took 2.2520 seconds
```



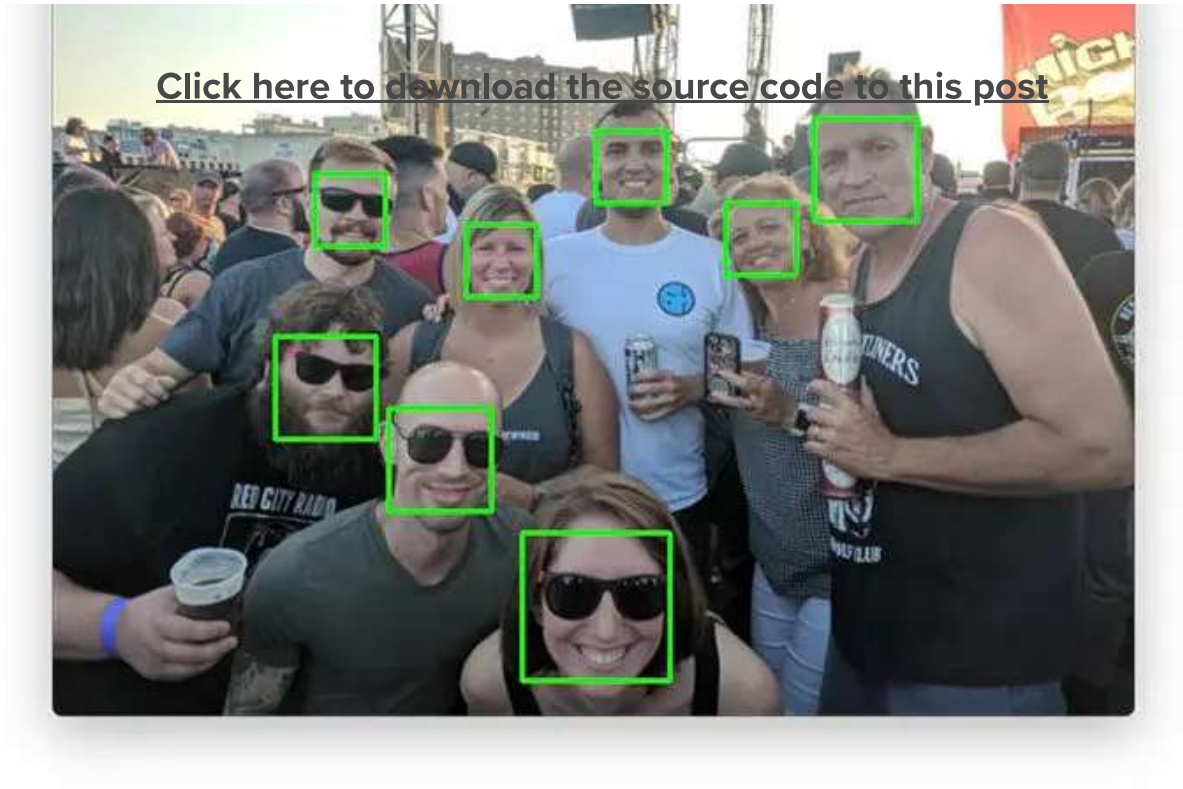


Figure 8: Dlib's deep learning face detector successfully detects all faces in the input image.

Before, using HOG + Linear SVM, we could only detect six of the eight faces in this image. But as our output shows, swapping over to dlib's deep learning face detector results in all eight faces being detected.

Which dlib face detector should I use?

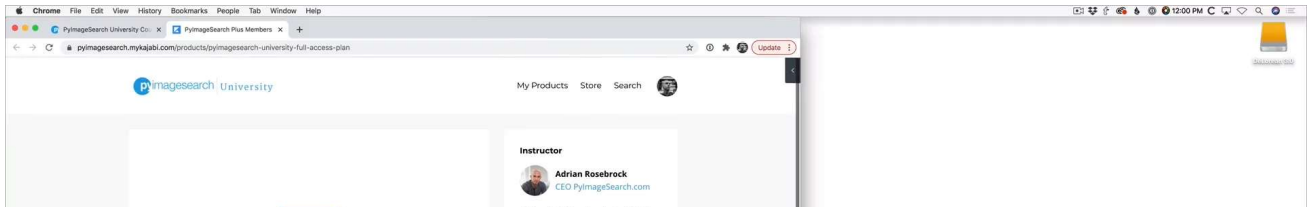
If you are using a CPU and speed is *not* an issue, use dlib's MMOD CNN face detector. It's *far* more accurate and robust than the HOG + Linear SVM face detector.

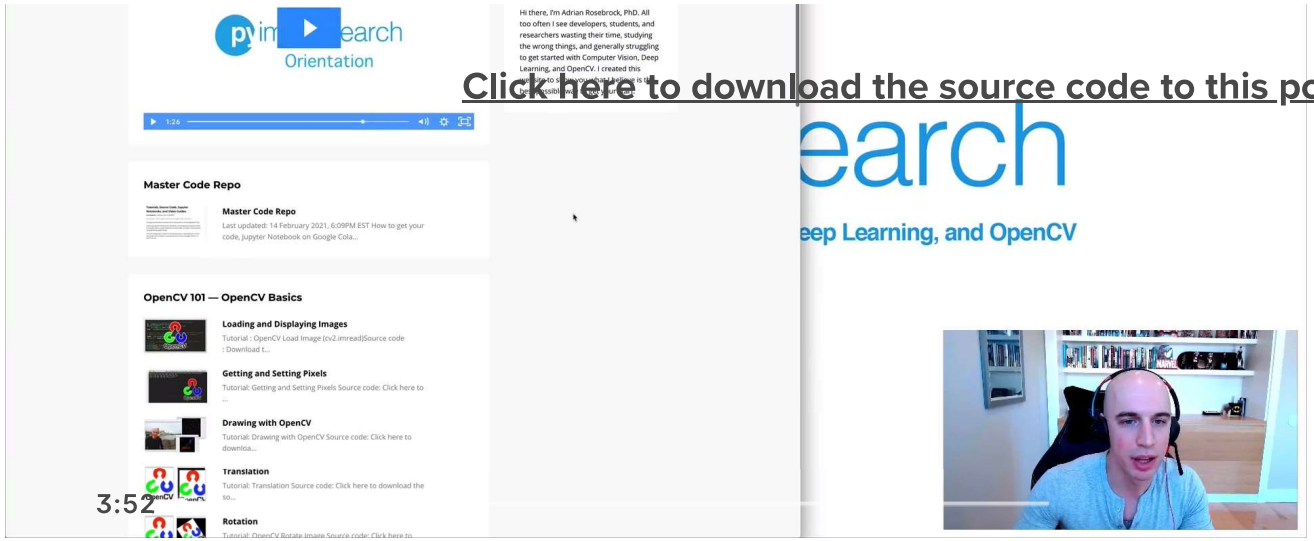
Additionally, if you have access to a GPU, then there's no doubt that you should be using the MMOD CNN face detector — you'll enjoy all the benefits of *accurate* face detection along with the *speed* of being able to run in real-time.

Click here to download the source code to this post

Suppose you are limited to just a CPU. In that case, speed *is* a concern, and you're willing to tolerate a bit less accuracy, then go with HOG + Linear SVM — it's still an accurate face detector and *significantly* more accurate than **OpenCV's Haar cascade face detector (https://www.pyimagesearch.com/2021/04/05/opencv-face-detection-with-haar-cascades/)**.

What's next? I recommend 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/?UTM_SOURCE=BLOGPOST&UTM_MEDIUM=BOTTOMBANNER&UTM_CAMPAIGN=WHAT%27S%20NEXT%3F%20I%20RECOMMEND)

Summary

In this tutorial, you learned how to perform face detection using the dlib library.

Dlib provides two methods to perform face detection:

1. **HOG + Linear SVM:** `dlib.get_frontal_face_detector()`
2. **MMOD CNN:** `dlib.cnn_face_detection_model_v1(modelPath)`

The HOG + Linear SVM face detector will be *faster* than the MMOD CNN face detector but will also be *less accurate* as HOG + Linear SVM does not tolerate changes in the viewing angle or rotation.

Click here to download the source code to this post

For more robust face detection, use dlib's MMOD CNN face detector. This model requires significantly more computation (and is thus slower) but is *much* more accurate and robust to changes in face rotation and viewing angle.

Furthermore, if you have access to a GPU, you can run dlib's MMOD CNN face detector on it, resulting in real-time face detection speed. The MMOD CNN face detector combined with a GPU is a match made in heaven — you get both the *accuracy* of a deep neural network along with the *speed* of a less computationally expensive model.

To download the source code to this post (and be notified when future tutorials are published here on PyImageSearch), *simply enter your email address in the form below!*

Download the Source Code and FREE 17-page Resource Guide

Click here to download the source code to this post

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

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.

Previous Article:

Your First Image Classifier: Using k-NN to Classify Images

(https://www.pyimagesearch.com/2021/04/17/your-first-image-classifier-using-k-nn-to-classify-images/)

Click here to download the source code to this post

Next Article:

Face detection tips, suggestions, and best practices

(https://www.pyimagesearch.com/2021/04/26/face-detection-tips-suggestions-and-best-practices/)

Comment section

Hey, Adrian Rosebrock here, author and creator of PyImageSearch. While I love hearing from readers, a couple years ago I made the tough decision to no longer offer 1:1 help over blog post comments.

At the time I was receiving 200+ emails per day and another 100+ blog post comments. I simply did not have the time to moderate and respond to them all, and the sheer volume of requests was taking a toll on me.

Instead, my goal is to *do the most good* for the computer vision, deep learning, and OpenCV community at large by focusing my time on authoring high-quality blog posts, tutorials, and books/courses.

If you need help learning computer vision and deep learning, I suggest you refer to my full catalog of books

If you need help learning computer vision and deep learning, I suggest you refer to my full catalog of books and courses (https://www.pyimagesearch.com/books-and-courses/) — they have helped tens of thousands of developers, students, and researchers *just like yourself* learn Computer Vision, Deep Learning, and OpenCV.

Click here to download the source code to this post

Click here to browse my full catalog. (https://www.pyimagesearch.com/books-and-courses/)

Similar articles

INTERVIEWS PYIMAGESEARCH GURUS

PyImageSearch Gurus member spotlight: Tuomo Hiippala

March 14, 2016

(https://www.pyimagesearch.com/2016/03/14/pyimagesearch-gurus-member-spotlight-tuomo-hiippala/)

Click here to download the source code to this post

IMAGE DESCRIPTORS TUTORIALS

OpenCV Shape Descriptor: Hu Moments Example

October 27, 2014

(https://www.pyimagesearch.com/2014/10/27/opencv-shape-descriptor-hu-moments-example/)

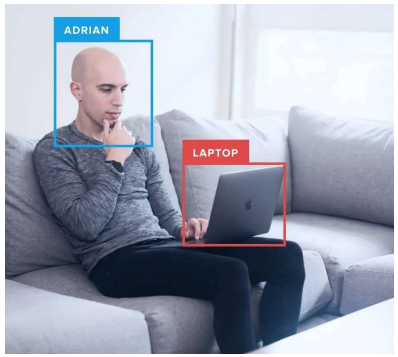
IMAGE PROCESSING TUTORIALS

Detecting Circles in Images using OpenCV and Hough Circles

July 21, 2014

(https://www.pyimagesearch.com/2014/07/21/detecting-circles-images-using-opencv-hough-circles/)

Click here to download the source code to this post



You can learn Computer Vision, Deep Learning, and OpenCV.

Get your FREE 17 page Computer Vision, OpenCV, and Deep Learning Resource Guide PDF. Inside you'll find our hand-picked tutorials, books, courses, and libraries to help you master CV and DL.

Topics

Deep Learning (https://www.pyimagesearch.com/category/deep-learning-2/)

Machine Learning and Computer Vision (https://www.pyimagesearch.com/category/machine-learning-2/)

Medical Computer Vision (https://www.pyimagesearch.com/category/medical/)

Dlib Library (https://www.pyimagesearch.com/category/dlib/)

Embedded/IoT and Computer Vision (https://www.pyimagesearch.com/category/embedded/)

Face Applications (https://www.pyimagesearch.com/category/faces/)

Image Processing (https://www.pyimagesearch.com/category/image-processing/)

Interviews (https://www.pyimagesearch.com/category/interviews/)

Keras (https://www.pyimagesearch.com/category/keras/)

Click here to download the source code to this post

Optical Character Recognition (OCR) (https://www.pyimagesearch.com/category/optical-character-recognition-ocr/)

Object Detection (https://www.pyimagesearch.com/category/object-detection/)

Object Tracking (https://www.pyimagesearch.com/category/object-tracking/)

OpenCV Tutorials (https://www.pyimagesearch.com/category/opencv/)

Raspberry Pi (https://www.pyimagesearch.com/category/raspberry-pi/)

Books & Courses

FREE CV, DL, and OpenCV Crash Course (https://www.pyimagesearch.com/free-opencv-computer-vision-deep-learning-crash-course/)
Practical Python and OpenCV (https://www.pyimagesearch.com/practical-python-opencv/)

Deep Learning for Computer Vision with Python (https://www.pyimagesearch.com/deep-learning-computer-vision-python-book/)

PyImageSearch Gurus Course (https://www.pyimagesearch.com/pyimagesearch-gurus/)

PyImageSearch

Get Started (https://www.pyimagesearch.com/start-here/)

OpenCV Install Guides (https://www.pyimagesearch.com/opencv-tutorials-resources-guides/)

About (https://www.pyimagesearch.com/about/)

FAQ (https://www.pyimagesearch.com/faqs/)

Blog (https://www.pyimagesearch.com/topics/)

Contact (https://www.pyimagesearch.com/contact/)

Privacy Policy (https://www.pyimagesearch.com/privacy-policy/)

Raspberry Pi for Computer Vision (https://www.pyimagesearch.com/raspberry-pi-for-computer-vision/)

Click here to download the source code to this post

(https://www.facebook.com/pyimagesearch) (https://twitter.com/PyImageSearch) (http://www.linkedin.com/pub/adrian-rosebrock/2a/873/59b) (https://www.youtube.com/channel/UCoQK7OVcIVy-nV4m-SMCk_Q/videos)