PYIMAGESEA
RCH

OpenCV Age Detection with Deep Learning

Click here to download the source code to this post (//app.monstercampaigns.com/c/tortsem7qkvyuxc4cyfi)

by **Adrian Rosebrock (https://www.pyimagesearch.com/author/adrian/)** *on* April 13, 2020

In this tutorial, you will learn how to perform automatic age detection/prediction using OpenCV, Deep Learning, and Python.

By the end of this tutorial, you will be able to automatically predict age in **static image files** and **real-time video streams** with reasonably high accuracy.

To learn how to perform age detection with OpenCV and Deep Learning, *just keep reading!*

Looking for the source code to this post?

JUMP RIGHT TO THE DOWNLOADS SECTION →

OpenCV Age Detection with Deep Learning

In the first part of this tutorial, you'll learn about age detection, including the steps required to automatically predict the age of a person from an image or a video stream (and why age detection is best treated as a *classification* problem rather than a *regression* problem).

From there, we'll discuss our deep learning-based age detection model and then learn how to use the model for both:

1. Age detection in static images
2. Age detection in real-time video streams

We'll then review the results of our age prediction work.

What is age detection?

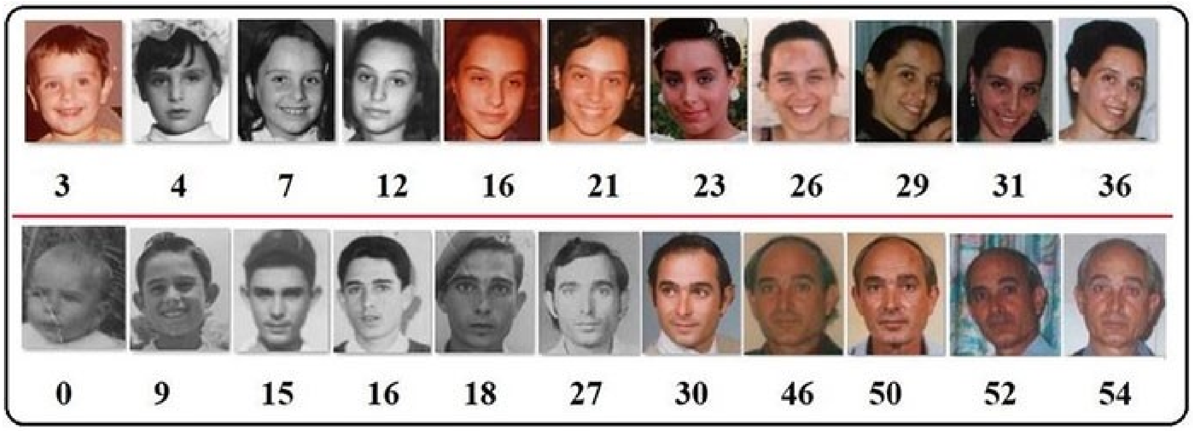


Figure 1: In this tutorial, we use OpenCV and a pre-trained deep learning model to predict the age of a given face (**image source (https://www.researchgate.net/publication/336955775_Age_Estimation_in_Make-up_Cosmetics_Using_Color_and_Depth_Images)**).

Age detection is the process of *automatically* discerning the age of a person *solely* from a photo of their face.

Typically, you'll see age detection implemented as a two-stage process:

1. **Stage #1:** Detect faces in the input image/video stream
2. **Stage #2:** Extract the face Region of Interest (ROI), and apply the age detector algorithm to predict the age of the person

For Stage #1, any face detector capable of producing bounding boxes for faces in an image can be used, including but not limited to Haar cascades, HOG + Linear SVM, Single Shot Detectors (SSDs), etc.

Exactly *which* face detector you use depends on your project:

- Haar cascades will be very fast and capable of running in real-time on embedded devices — the *problem* is that they are less accurate and highly prone to false-positive detections
- HOG + Linear SVM models are more accurate than Haar cascades but are slower. They also aren't as tolerant with occlusion (i.e., not all of the face visible) or viewpoint changes (i.e., different views of the face)
- Deep learning-based face detectors are the most robust and will give you the best accuracy, but require even more computational resources than both Haar cascades and HOG + Linear SVMs

the empirical results guide your decisions.

Once your face detector has produced the bounding box coordinates of the face in the image/video stream, you can move on to Stage #2 — identifying the age of the person.

Given the bounding box *(x, y)*-coordinates of the face, you first extract the face ROI, ignoring the rest of the image/frame. Doing so allows the age detector to focus *solely* on the person's face and not any other irrelevant "noise" in the image.

The face ROI is then passed through the model, yielding the actual age prediction.

There are a number of age detector algorithms, but the most popular ones are deep learning-based age detectors — we'll be using such a deep learning-based age detector in this tutorial.

Our age detector deep learning model

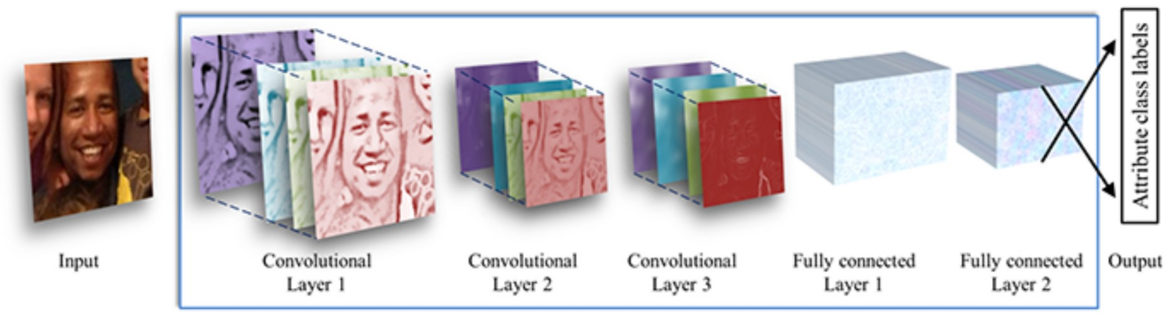


Figure 2: Deep learning age detection is an active area of research. In this tutorial, we use the model implemented and trained by Levi and Hassner in their 2015 paper (**image source, Figure 2 (https://talhassner.github.io/home/projects/cnn_agegender/CVPR2015_CNN_AgeGenderEstimation.pdf)**).

The deep learning age detector model we are using here today was implemented and trained by Levi and Hassner in their 2015 publication, ***Age and Gender Classification***

total of eight age brackets:

1. 0-2
2. 4-6
3. 8-12
4. 15-20
5. 25-32
6. 38-43
7. 48-53
8. 60-100

You'll note that these age brackets are *noncontiguous* — this done on purpose, as the **Adience dataset (https://talhassner.github.io/home/projects/Adience/Adience-data.html#agegender)**, used to train the model, defines the age ranges as such (we'll learn why this is done in the next section).

We'll be using a *pre-trained* age detector model in this post, but if you are interested in learning how to train it from scratch, be sure to read ***Deep Learning for Computer Vision with Python (https://www.pyimagesearch.com/deep-learning-computer-vision-python-book/)***, where I show you how to do exactly that.

Why aren't we treating age prediction as a regression problem?

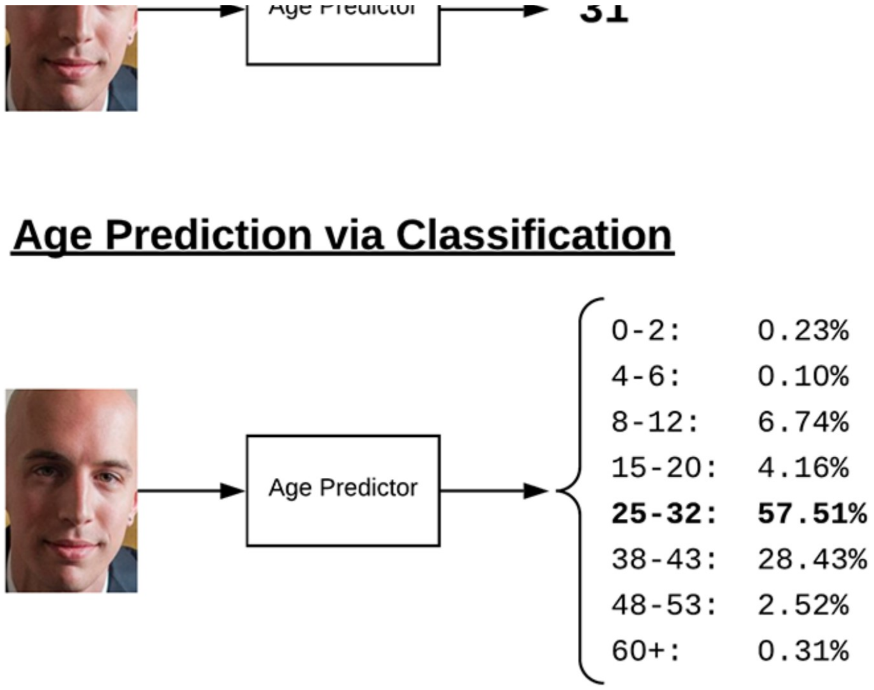


Figure 3: Age prediction with deep learning can be framed as a regression or classification problem.

You'll notice from the previous section that we have discretized ages into "buckets," thereby treating age prediction as a ***classification*** problem — **why not frame it as a *regression* problem instead** (the way we did in our **house price prediction tutorial) (https://www.pyimagesearch.com/2019/01/28/keras-regression-and-cnns/)**?

Technically, there's no reason why you can't treat age prediction as a regression task. There are even some models that do just that.

The problem is that age prediction is inherently subjective and based *solely* on appearance.

A person in their mid-50s who has never smoked in their life, always wore sunscreen when going outside, and took care of their skin daily will likely look younger than someone in their late-30s who smokes a carton a day, works manual labor without sun

For example, take a look at the following image of Matthew Perry (who played Chandler Bing on the TV sitcom, *Friends*) and compare it to an image of Jennifer Aniston (who played Rachel Green, alongside Perry):

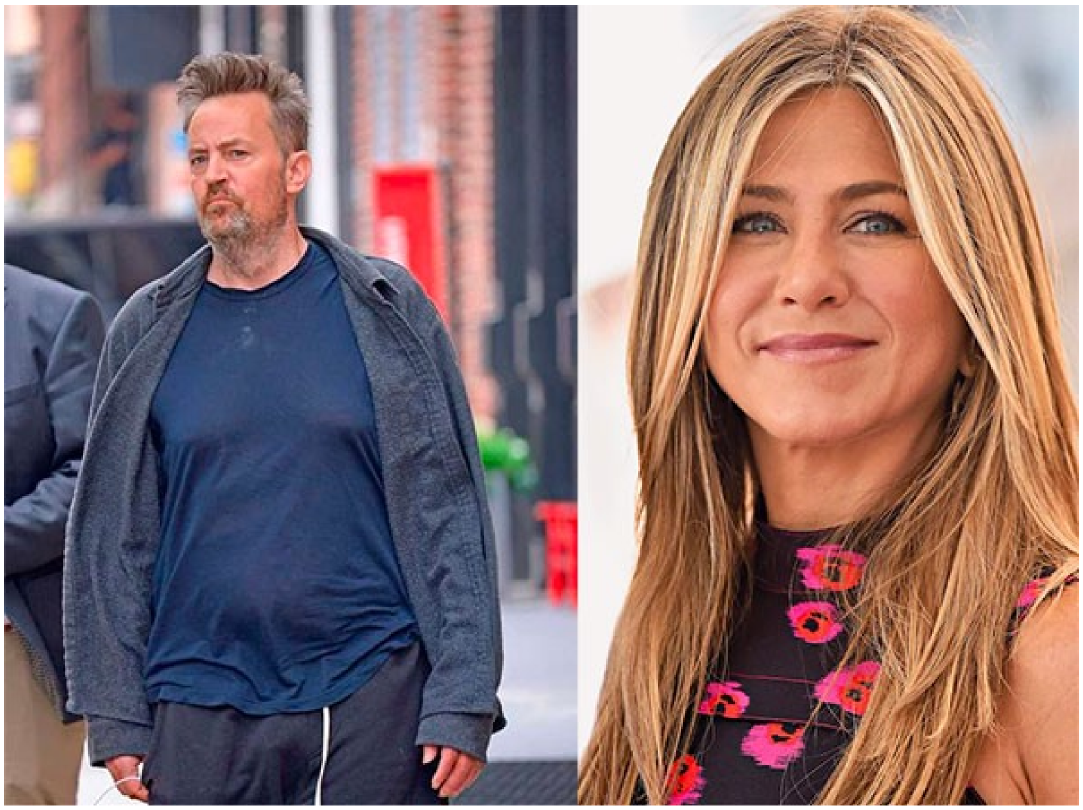


Figure 4: Many celebrities and figure heads work hard to make themselves look younger. This presents a challenge for deep learning age detection with OpenCV.

Could you guess that Matthew Perry (50) is actually a year younger than Jennifer Aniston (51)?

Unless you have prior knowledge about these actors, *I doubt it*.

But, on the other hand, could you guess that these actors were 48-53?

I'm willing to bet you probably could.

Jennifer Aniston's genetics are near perfect, and combined with an extremely talented plastic surgeon, she seems to never age.

But that goes to show my point — people *purposely* try to hide their age.

And if a *human* struggles to accurately predict the age of a person, then surely a *machine* will struggle as well.

Once you start treating age prediction as a regression problem, it becomes *significantly* harder for a model to accurately predict a single value representing that person's image.

However, if you treat it as a classification problem, defining buckets/age brackets for the model, our age predictor model becomes easier to train, often yielding substantially higher accuracy than regression-based prediction alone.

Simply put: **Treating age prediction as classification "relaxes" the problem a bit, making it easier to solve — typically, we don't need the *exact* age of a person; a rough estimate is sufficient.**

Project structure

Be sure to grab the code, models, and images from the ***"Downloads"*** section of this blog post. Once you extract the files, your project will look like this:

OpenCV Age Detection with Deep Learning

```
$ tree --dirsfirst
.
├── age_detector
│   ├── age_deploy.prototxt
│   └── age_net.caffemodel
├── face_detector
│   ├── deploy.prototxt
│   └── res10_300x300_ssd_iter_140000.caffemodel
├── images
```

```
3 directories, 9 files
```

The first two directories consist of our **age predictor** and **face detector**. Each of these deep learning models is Caffe-based.

I've provided three testing images for age prediction; you can add your own images as well.

In the remainder of this tutorial, we will review two Python scripts:

- `detect_age.py` : Single image age prediction
- `detect_age_video.py` : Age prediction in video streams

Each of these scripts detects faces in an image/frame and then performs age prediction on them using OpenCV.

Implementing our OpenCV age detector for images

Let's get started by implementing age detection with OpenCV in static images.

Open up the `detect_age.py` file in your project directory, and let's get to work:

OpenCV Age Detection with Deep Learning

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

# 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("-f", "--face", required=True,
	help="path to face detector model directory")
ap.add_argument("-a", "--age", required=True,
	help="path to age detector model directory")
```

my ***pip install opencv (https://www.pyimagesearch.com/2018/09/19/pip-install-opencv/)*** tutorial to configure your system.

Additionally, we need to import Python's built-in `os` module for joining our model paths.

And finally, we **import argparse to parse command line arguments (https://www.pyimagesearch.com/2018/03/12/python-argparse-command-line-arguments/)**.

Our script requires four command line arguments:

- `--image` : Provides the path to the input image for age detection
- `--face` : The path to our pre-trained face detector model directory
- `--age` : Our pre-trained age detector model directory
- `--confidence` : The minimum probability threshold in order to filter weak detections

As we learned above, our age detector is a classifier that predicts a person's age using their face ROI according to predefined buckets — we aren't treating this as a regression problem. Let's define those age range buckets now:

OpenCV Age Detection with Deep Learning

```
# define the list of age buckets our age detector will predict
AGE_BUCKETS = ["(0-2)", "(4-6)", "(8-12)", "(15-20)", "(25-32)",
	"(38-43)", "(48-53)", "(60-100)"]
```

Our ages are defined in buckets (i.e., class labels) for our pre-trained age detector. We'll use this list and an associated index to grab the age bucket to annotate on the output image.

Given our imports, command line arguments, and age buckets, we're now ready to load our two pre-trained models:

```
28. faceNet = cv2.dnn.readNet(prototxtPath, weightsPath)
29.
30. # load our serialized age detector model from disk
31. print("[INFO] loading age detector model...")
32. prototxtPath = os.path.sep.join([args["age"], "age_deploy.prototxt"])
33. weightsPath = os.path.sep.join([args["age"], "age_net.caffemodel"])
34. ageNet = cv2.dnn.readNet(prototxtPath, weightsPath)
```

Here, we load two models:

- Our face detector finds and localizes faces in the image (**Lines 25-28**)
- The age classifier determines which age range a particular face belongs to (**Lines 32-34**)

Each of these models was trained with the Caffe framework. I cover how to train Caffe classifiers inside the ***PyImageSearch Gurus (https://www.pyimagesearch.com/pyimagesearch-gurus/)* course (https://www.pyimagesearch.com/pyimagesearch-gurus/)**.

Now that all of our initializations are taken care of, let's load an image from disk and detect face ROIs:

```
OpenCV Age Detection with Deep Learning
36. # load the input image and construct an input blob for the image
37. image = cv2.imread(args["image"])
38. (h, w) = image.shape[:2]
39. blob = cv2.dnn.blobFromImage(image, 1.0, (300, 300),
40.     (104.0, 177.0, 123.0))
41.
42. # pass the blob through the network and obtain the face detections
43. print("[INFO] computing face detections...")
44. faceNet.setInput(blob)
45. detections = faceNet.forward()
```

Lines 37-40 load and preprocess our input `--image`. We use OpenCV's `blobFromImage` method — be sure to **read more about blobFromImage in my tutorial (https://www.pyimagesearch.com/2017/11/06/deep-learning-opencvs-blobfromimage-**

OpenCV Age Detection with Deep Learning

```
47.     # loop over the detections
48.     for i in range(0, detections.shape[2]):
49.         # extract the confidence (i.e., probability) associated with the
50.         # prediction
51.         confidence = detections[0, 0, i, 2]
52.
53.         # filter out weak detections by ensuring the confidence is
54.         # greater than the minimum confidence
55.         if confidence > args["confidence"]:
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.             # extract the ROI of the face and then construct a blob from
62.             # *only* the face ROI
63.             face = image[startY:endY, startX:endX]
64.             faceBlob = cv2.dnn.blobFromImage(face, 1.0, (227, 227),
65.                 (78.4263377603, 87.7689143744, 114.895847746),
66.                 swapRB=False)
```

As we loop over the `detections`, we filter out weak `confidence` faces (**Lines 51-55**).

For faces that meet the minimum confidence criteria, we extract the ROI coordinates (**Lines 58-63**). At this point, we have a small crop from the image containing only a face. We go ahead and create a blob from this ROI (i.e., `faceBlob`) via **Lines 64-66**.

And now we'll **perform age detection:**

OpenCV Age Detection with Deep Learning

```
68.             # make predictions on the age and find the age bucket with
69.             # the largest corresponding probability
70.             ageNet.setInput(faceBlob)
71.             preds = ageNet.forward()
72.             i = preds[0].argmax()
73.             age = AGE_BUCKETS[i]
74.             ageConfidence = preds[0][i]
75.
76.             # display the predicted age to our terminal
77.             text = "{}: {:.2f}%".format(age, ageConfidence * 100)
78.             print("[INFO] {}".format(text))
79.
80.             # draw the bounding box of the face along with the associated
```

```
87.
88.  # display the output image
89.  cv2.imshow("Image", image)
90.  cv2.waitKey(0)
```

Using our face blob, we make age predictions (**Lines 70-74**) resulting in an `age` bucket and `ageConfidence` . We use these data points along with the coordinates of the face ROI to annotate the original input `--image` (**Lines 77-86**) and display results (**Lines 89 and 90**).

In the next section, we'll analyze our results.

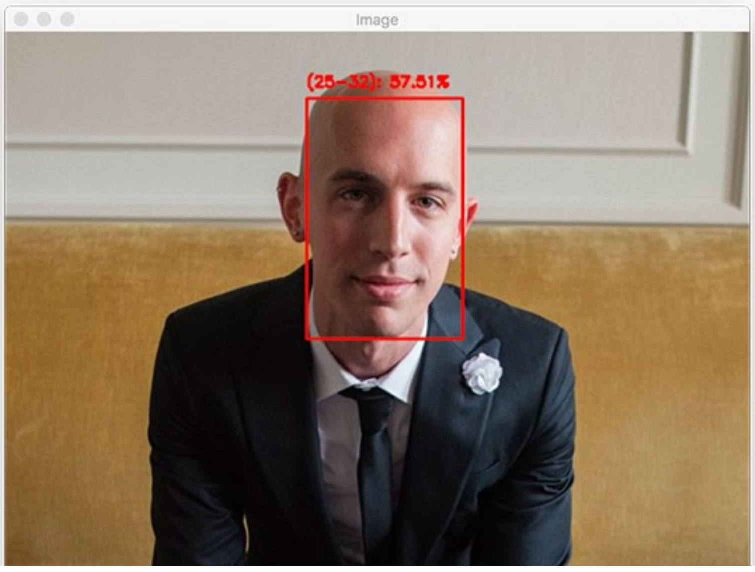
OpenCV age detection results

Let's put our OpenCV age detector to work.

Start by using the ***"Downloads"*** section of this tutorial to download the source code, pre-trained age detector model, and example images.

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

```
OpenCV Age Detection with Deep Learning
$ python detect_age.py --image images/adrian.png --face face_detector --age age_detector
[INFO] loading face detector model...
[INFO] loading age detector model...
[INFO] computing face detections...
[INFO] (25-32): 57.51%
```



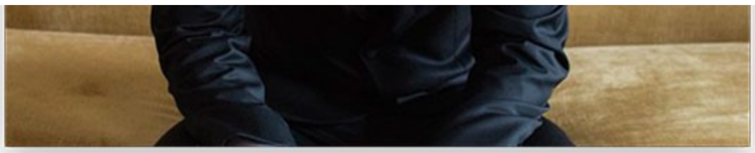


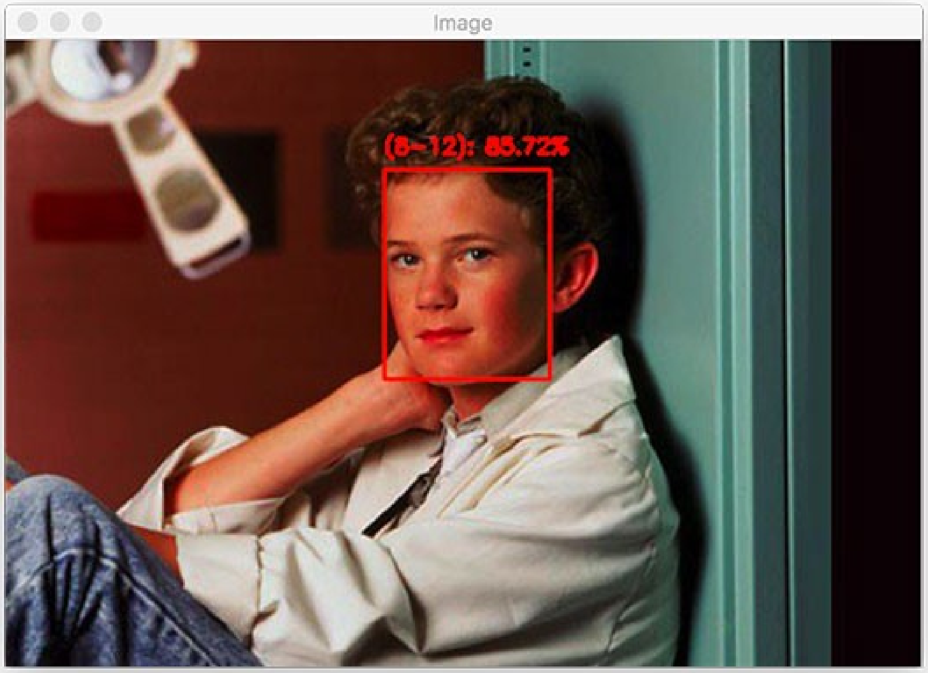
Figure 5: Age detection with OpenCV has correctly identified me in this photo of me when I was 30 years old.

Here, you can see that our OpenCV age detector has predicted my age to be 25-32 with 57.51% confidence — indeed, the age detector is *correct* (I was 30 when that picture was taken).

Let's try another example, this one of the famous actor, Neil Patrick Harris when he was a kid:

OpenCV Age Detection with Deep Learning

```
$ python detect_age.py --image images/neil_patrick_harris.png --face face_detector --age age_detector
[INFO] loading face detector model...
[INFO] loading age detector model...
[INFO] computing face detections...
[INFO] (8-12): 85.72%
```



Harris was 8-12 years old when this photo was taken.

Our age predictor is one again *correct* — Neil Patrick Harris certainly looked to be somewhere in the 8-12 age group when this photo was taken.

Let’s try another image; this image is of one of my favorite actors, the infamous Samuel L. Jackson:

OpenCV Age Detection with Deep Learning

```
$ python detect_age.py --image images/samuel_l_jackson.png --face face_detector --age age_detector
[INFO] loading face detector model...
[INFO] loading age detector model...
[INFO] computing face detections...
[INFO] (48-53): 69.38%
```

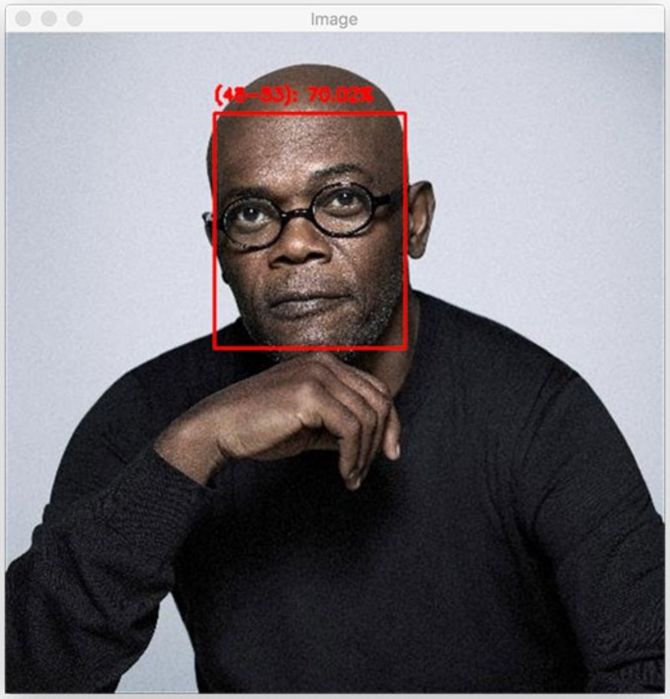


Figure 7: Deep learning age prediction with OpenCV isn’t always accurate, as is evident in this photo of Samuel L. Jackson. Age prediction is subjective for humans just as it is for

That said, look at the photo — does Mr. Jackson actually *look* to be 71?

My guess would have been late 50s to early 60s. At least to me, he certainly doesn't look like a man in his early 70s.

But that just goes to show my point earlier in this post:

The process of visual age prediction is difficult, and I'd consider it *subjective* when either a computer or a person tries to guess someone's age.

In order to evaluate an age detector, you cannot rely on the person's actual age. Instead, you need to measure the accuracy between the *predicted* age and the *perceived* age.

Implementing our OpenCV age detector for real-time video streams

At this point, we can perform age detection in static images, but what about real-time video streams?

Can we do that as well?

You bet we can. Our video script very closely aligns with our image script. The difference is that we need to set up a video stream and perform age detection on each and every frame in a loop. This review will focus on the video features, so be sure to refer to the walkthrough above as needed.

To see how to perform age recognition in video, let's take a look at `detect_age_video.py`.

OpenCV Age Detection with Deep Learning

```
# import the necessary packages
from imutils.video import VideoStream
```

We have three new imports: (1) `VideoStream`, (2) `imutils`, and (3) `time`. Each of these imports allow us to set up and use our webcam for our video stream.

I've decided to define a convenience function for accepting a `frame`, localizing faces, and predicting ages. By putting the detect and predict logic here, our frame processing loop will be less bloated (you could also offload this function to a separate file). Let's dive into this utility now:

OpenCV Age Detection with Deep Learning

```
def detect_and_predict_age(frame, faceNet, ageNet, minConf=0.5):
    # define the list of age buckets our age detector will predict
    AGE_BUCKETS = ["(0-2)", "(4-6)", "(8-12)", "(15-20)", "(25-32)",
        "(38-43)", "(48-53)", "(60-100)"]

    # initialize our results list
    results = []

    # grab the dimensions of the frame and then construct a blob
    # from it
    (h, w) = frame.shape[:2]
    blob = cv2.dnn.blobFromImage(frame, 1.0, (300, 300),
        (104.0, 177.0, 123.0))

    # pass the blob through the network and obtain the face detections
    faceNet.setInput(blob)
    detections = faceNet.forward()
```

Our `detect_and_predict_age` helper function accepts the following parameters:

- `frame` : A single frame from your webcam video stream
- `faceNet` : The initialized deep learning face detector
- `ageNet` : Our initialized deep learning age classifier
- `minConf` : The confidence threshold to filter weak face detections

These parameters draw parallels from the command line arguments of our single image

We then initialize an empty list to hold the `results` of face localization and age detection.

Lines 20-26 handle **performing face detection**.

Next, we'll process each of the `detections` :

OpenCV Age Detection with Deep Learning

```
28.     # loop over the detections
29.     for i in range(0, detections.shape[2]):
30.         # extract the confidence (i.e., probability) associated with
31.         # the prediction
32.         confidence = detections[0, 0, i, 2]
33. 
34.         # filter out weak detections by ensuring the confidence is
35.         # greater than the minimum confidence
36.         if confidence > minConf:
37.             # compute the (x, y)-coordinates of the bounding box for
38.             # the object
39.             box = detections[0, 0, i, 3:7] * np.array([w, h, w, h])
40.             (startX, startY, endX, endY) = box.astype("int")
41. 
42.             # extract the ROI of the face
43.             face = frame[startY:endY, startX:endX]
44. 
45.             # ensure the face ROI is sufficiently large
46.             if face.shape[0] < 20 or face.shape[1] < 20:
47.                 continue
```

You should recognize **Lines 29-43** — they loop over `detections` , ensure high `confidence` , and extract a `face` ROI.

Lines 46 and 47 are new — they ensure that a face ROI is sufficiently large in our stream for two reasons:

- First, we want to filter out false-positive face detections in the frame.
- Second, age classification results won't be accurate for faces that are far away from the camera (i.e., perceivably small).

```
51.                   (78.4263377603, 87.7689143744, 114.895847746),
52.                   swapRB=False)
53.
54.               # make predictions on the age and find the age bucket with
55.               # the largest corresponding probability
56.               ageNet.setInput(faceBlob)
57.               preds = ageNet.forward()
58.               i = preds[0].argmax()
59.               age = AGE_BUCKETS[i]
60.               ageConfidence = preds[0][i]
61.
62.               # construct a dictionary consisting of both the face
63.               # bounding box location along with the age prediction,
64.               # then update our results list
65.               d = {
66.                   "loc": (startX, startY, endX, endY),
67.                   "age": (age, ageConfidence)
68.               }
69.               results.append(d)
70.
71.       # return our results to the calling function
72.       return results
```

Here, we predict the age of the face and extract the `age` bucket and `ageConfidence` (**Lines 56-60**).

Lines 65-68 arrange face localization and predicted age in a dictionary. The last step of the detection processing loop is to add the dictionary to the `results` list (**Line 69**).

Once all `detections` have been processed and any `results` are ready, we `return` the results to the caller.

With our helper function defined, now we can get back to working with our video stream. But first, we need to define command line arguments:

OpenCV Age Detection with Deep Learning

```
74.  # construct the argument parse and parse the arguments
75.  ap = argparse.ArgumentParser()
76.  ap.add_argument("-f", "--face", required=True,
77.     help="path to face detector model directory")
78.  ap.add_argument("-a", "--age", required=True,
79.     help="path to age detector model directory")
80.  ap.add_argument("-c", "--confidence", type=float, default=0.5,
```

/2018/03/12/python-argparse-command-line-arguments/).

- `--face` : The path to our pre-trained face detector model directory
- `--age` : Our pre-trained age detector model directory
- `--confidence` : The minimum probability threshold in order to filter weak detections

From here, we'll load our models and initialize our video stream:

```
      OpenCV Age Detection with Deep Learning
 84. # load our serialized face detector model from disk
 85. print("[INFO] loading face detector model...")
 86. prototxtPath = os.path.sep.join([args["face"], "deploy.prototxt"])
 87. weightsPath = os.path.sep.join([args["face"],
 88.     "res10_300x300_ssd_iter_140000.caffemodel"])
 89. faceNet = cv2.dnn.readNet(prototxtPath, weightsPath)
 90.
 91. # load our serialized age detector model from disk
 92. print("[INFO] loading age detector model...")
 93. prototxtPath = os.path.sep.join([args["age"], "age_deploy.prototxt"])
 94. weightsPath = os.path.sep.join([args["age"], "age_net.caffemodel"])
 95. ageNet = cv2.dnn.readNet(prototxtPath, weightsPath)
 96.
 97. # initialize the video stream and allow the camera sensor to warm up
 98. print("[INFO] starting video stream...")
 99. vs = VideoStream(src=0).start()
100. time.sleep(2.0)
```

Lines 86-89 load and initialize our face detector, while **Lines 93-95** load our age detector.

We then use the `VideoStream` class to initialize our webcam (**Lines 99 and 100**).

Once our webcam is warmed up, we'll begin processing frames:

```
      OpenCV Age Detection with Deep Learning
102. # loop over the frames from the video stream
103. while True:
104.     # grab the frame from the threaded video stream and resize it
105.     # to have a maximum width of 400 pixels
106.     frame = vs.read()
```

```
113. 
114.     # loop over the results
115.     for r in results:
116.         # draw the bounding box of the face along with the associated
117.         # predicted age
118.         text = "{}: {:.2f}%".format(r["age"][0], r["age"][1] * 100)
119.         (startX, startY, endX, endY) = r["loc"]
120.         y = startY - 10 if startY - 10 > 10 else startY + 10
121.         cv2.rectangle(frame, (startX, startY), (endX, endY),
122.             (0, 0, 255), 2)
123.         cv2.putText(frame, text, (startX, y),
124.             cv2.FONT_HERSHEY_SIMPLEX, 0.45, (0, 0, 255), 2)
125. 
126.     # show the output frame
127.     cv2.imshow("Frame", frame)
128.     key = cv2.waitKey(1) & 0xFF
129. 
130.     # if the `q` key was pressed, break from the loop
131.     if key == ord("q"):
132.         break
133. 
134. # do a bit of cleanup
135. cv2.destroyAllWindows()
136. vs.stop()
```

Inside our loop, we:

- Grab the next `frame`, and resize it to a known width (**Lines 106 and 107**)
- Send the `frame` through our `detect_and_predict_age` convenience function to (1) detect faces and (2) determine ages (**Lines 111 and 112**)
- Annotate the `results` on the `frame` (**Lines 115-124**)
- Display and capture keypresses (**Lines 127 and 128**)
- Exit and clean up if the `q` key was pressed (**Lines 131-136**)

In the next section, we'll fire up our age detector and see if it works!

Real-time age detection with OpenCV results

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

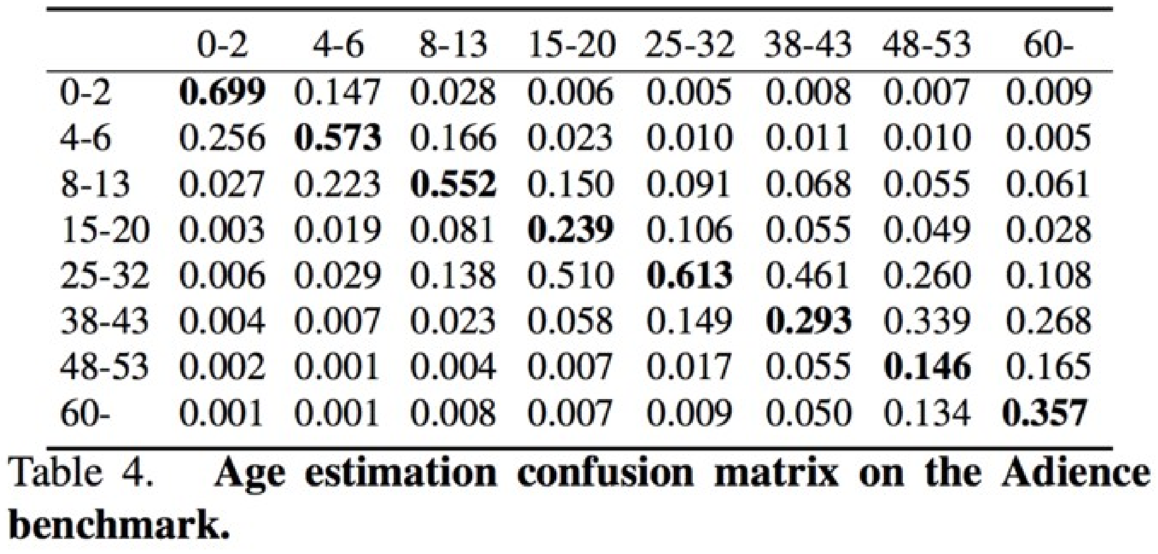
OpenCV Age Detection with Deep Learning

```
$ python detect_age_video.py --face face_detector --age age_detector
[INFO] loading face detector model...
[INFO] loading age detector model...
[INFO] starting video stream...
```

Here, you can see that our OpenCV age detector is accurately predicting my age range as 25-32 (I am currently 31 at the time of this writing).

How can I improve age prediction results?

/home/projects/cnn_agegender/CVPR2015_CNN_AgeGenderEstimation.pdf).



	0-2	4-6	8-13	15-20	25-32	38-43	48-53	60-
0-2	**0.699**	0.147	0.028	0.006	0.005	0.008	0.007	0.009
4-6	0.256	**0.573**	0.166	0.023	0.010	0.011	0.010	0.005
8-13	0.027	0.223	**0.552**	0.150	0.091	0.068	0.055	0.061
15-20	0.003	0.019	0.081	**0.239**	0.106	0.055	0.049	0.028
25-32	0.006	0.029	0.138	0.510	**0.613**	0.461	0.260	0.108
38-43	0.004	0.007	0.023	0.058	0.149	**0.293**	0.339	0.268
48-53	0.002	0.001	0.004	0.007	0.017	0.055	**0.146**	0.165
60-	0.001	0.001	0.008	0.007	0.009	0.050	0.134	**0.357**

Table 4. **Age estimation confusion matrix on the Adience benchmark.**

Figure 8: The Levi and Hassner deep learning age detection model is heavily biased toward the age range 25-32. To combat this in your own models, consider gathering more training data, applying class weighting, data augmentation, and regularization techniques. (image source: **Table 4 (https://talhassner.github.io/home/projects/cnn_agegender/CVPR2015_CNN_AgeGenderEstimation.pdf)**)

That unfortunately means that our model may predict the 25-32 age group when in fact the actual age belongs to a different age bracket — I noticed this a handful of times when gathering results for this tutorial as well as in my own applications of age prediction.

You can combat this bias by:

1. Gathering additional training data for the other age groups to help balance out the dataset
2. Applying class weighting to handle class imbalance
3. Being more aggressive with data augmentation

(https://www.pyimagesearch.com/2017/05/22/face-alignment-with-opencv-and-python/)

Face alignment identifies the geometric structure of faces and then attempts to obtain a canonical alignment of the face based on translation, scale, and rotation.

In many cases (but not always), face alignment can improve face application results, including face recognition, age prediction, etc.

As a matter of simplicity, we did *not* apply face alignment in this tutorial, but you can **follow this tutorial (https://www.pyimagesearch.com/2017/05/22/face-alignment-with-opencv-and-python/)** to learn more about face alignment and then apply it to your own age prediction applications.

What about gender prediction?

I have chosen to *purposely* not cover gender prediction in this tutorial.

While using computer vision and deep learning to identify the gender of a person may seem like an interesting classification problem, it's actually one wrought with *moral implications.*

Just because someone visually *looks, dresses,* or *appears* a certain way ***does not*** imply they identify with that (or any) gender.

Software that attempts to distill gender into binary classification only further chains us to antiquated notions of what gender is. Therefore, I would encourage you to *not* utilize gender recognition in your own applications if at all possible.

If you *must* perform gender recognition, make sure you are holding yourself accountable, and ensure you are not building applications that attempt to conform solves. Try to avoid it if at all possible.

Do you want to train your own deep learning models?

(https://www.pyimagesearch.com/deep-learning-computer-vision-python-book/)

Figure 9: Pick up a copy of ***Deep Learning for Computer Vision with Python* (https://www.pyimagesearch.com/deep-learning-computer-vision-python-book/)** to learn how to train your own deep learning models, including an age detector.

In the blog post, I showed you how to use a *pre-trained* age detector — if you instead want to learn how to *train* the age detector from scratch, I would recommend you check out my book, ***Deep Learning for Computer Vision with Python* *(https://www.pyimagesearch.com/deep-learning-computer-vision-python-book/)***.

Inside the book, you'll find:

- **Super-practical walkthroughs** that present solutions to actual real-world image classification (ResNet, VGG, etc.), object detection (Faster R-CNN, SSDs, RetinaNet, etc.), and segmentation (Mask R-CNN) problems
- **Hands-on tutorials (with lots of code)** that show you not only the *algorithms* behind

If you're interested in learning more about the book, I'd be happy to send you a PDF containing the Table of Contents and a few sample chapters:

Send me the *free* chapters! (https://www.pyimagesearch.com/deep-learning-computer-vision-python-book)

Summary

In this tutorial, you learned how to perform age detection with OpenCV and Deep Learning.

To do so, we utilized a pre-trained model from Levi and Hassner in their 2015 publication, *Age and Gender Classification using Convolutional Neural Networks.* This model allowed us to predict eight different age groups with reasonably high accuracy; however, we must recognize that age prediction is a *challenging problem.*

There are a number of factors that determine how old a person *visually appears*, including their lifestyle, work/job, smoking habits, and most importantly, *genetics.* Secondly, keep in mind that people *purposely* try to hide their age — if a human struggles to accurately predict someone's age, then surely a machine learning model will struggle as well.

Therefore, you must assess all age prediction results in terms of *perceived age* rather than *actual age.* Keep this in mind when implementing age detection into your own computer vision projects.

I hope you enjoyed this tutorial!