Face Detection Models: Which to Use and Why?

A complete tutorial on implementing different face detection models in Python followed, by comparison, to find out the best one to use for real-time scenarios.

Vardan Agarwal Jul 2, 2020 · 11 min read ★

Modified photo by Jerry Zhang on Unsplash.

Face detection is one of the most fundamental aspects of computer vision. It is the base of many further studies like identifying specific people to marking key points on the face. Recently, it has been quite a lot in the news due to racial profiling incidents as elaborated here and here where people of color are being misidentified more than white people. So major tech companies like IBM, Microsoft, and Amazon have banned their systems being used by the police. However, this article won't be dwelling on those aspects and we will merely be trying to draw bounding boxes on faces using pre-trained models like Haar cascades, dlib frontal face detector, MTCNN, and a Caffe model using OpenCV's DNN module. Then we will compare them to find out which works the best for real-time applications.

Table of Contents

Introduction

We will be using Haar, dlib, Multi-task Cascaded Convolutional Neural Network (MTCNN), and OpenCV's DNN module. If you already know about them or don't want to go in their technical details, feel free to skip this section and move straight on to the code. Otherwise, let's learn how they work.

Haar Cascades

They were proposed way back in 2001 by Paul Viola and Micheal Jones in their paper, "Rapid Object Detection using a Boosted Cascade of Simple Features." It is super fast to work with and like the simple CNN, it extracts a lot of features from images. The best features are then selected via Adaboost. This reduces the original 160000+ features to 6000 features. But applying all these features in a sliding window will still take a lot of time. So they introduced a Cascade of Classifiers, where the features are grouped. If a window fails at the first stage, these remaining features in that cascade are not processed. If it passes then the next feature is tested and the same procedure is repeated. If a window can pass all the features then it is classified as a face region. For more detailed reading, you can refer here.

Haar cascades require a lot of positive and negative training images to train. Thankfully, these cascades come bundled with the OpenCV library along with the trained XML files.

Dlib Frontal Face Detector

Dlib is a C++ toolkit containing machine learning algorithms used to solve real-world problems. Although it is written in C++ it has python bindings to run it in python. It also has the great facial landmark keypoint detector which I used in one of my earlier articles to make a real-time gaze tracking system.

Real-time eye tracking using OpenCV and Dlib

Learn to create a real-time gaze detector through the webcam in python with this tutorial.

towardsdatascience.com

The frontal face detector provided by dlib works using features extracted by Histogram of Oriented Gradients (HOG) which are then passed through an SVM. In the HOG feature descriptor, the distribution of the directions of gradients is used as features. Moreover, Dlib provides a more advanced CNN based face detector, however, that does not work in real-time on a CPU which is one of the goals we are looking for so it has been ignored in this article. Nonetheless, if you want to read about it you can refer here.

MTCNN

It was introduced by Kaipeng Zhang, et al. in 2016 in their paper, "Joint Face Detection and Alignment Using Multi-task Cascaded Convolutional Networks." It not only detects the face but also detects five key points as well. It uses a cascade structure with three stages of CNN. First, they use a fully convolutional network to obtain candidate windows and their bounding box regression vectors, and the highly overlapped candidates are overlapped using on-maximum suppression (NMS). Next, these candidates are passed to another CNN which rejects a large number of false positives and performs calibration of bounding boxes. In the final stage, the facial landmark detection is performed.

DNN Face Detector in OpenCV

It is a Caffe model which is based on the Single Shot-Multibox Detector (SSD) and uses ResNet-10 architecture as its backbone. It was introduced post OpenCV 3.3 in its deep neural network module. There is also a quantized Tensorflow version that can be used but we will use the Caffe Model.

Installation

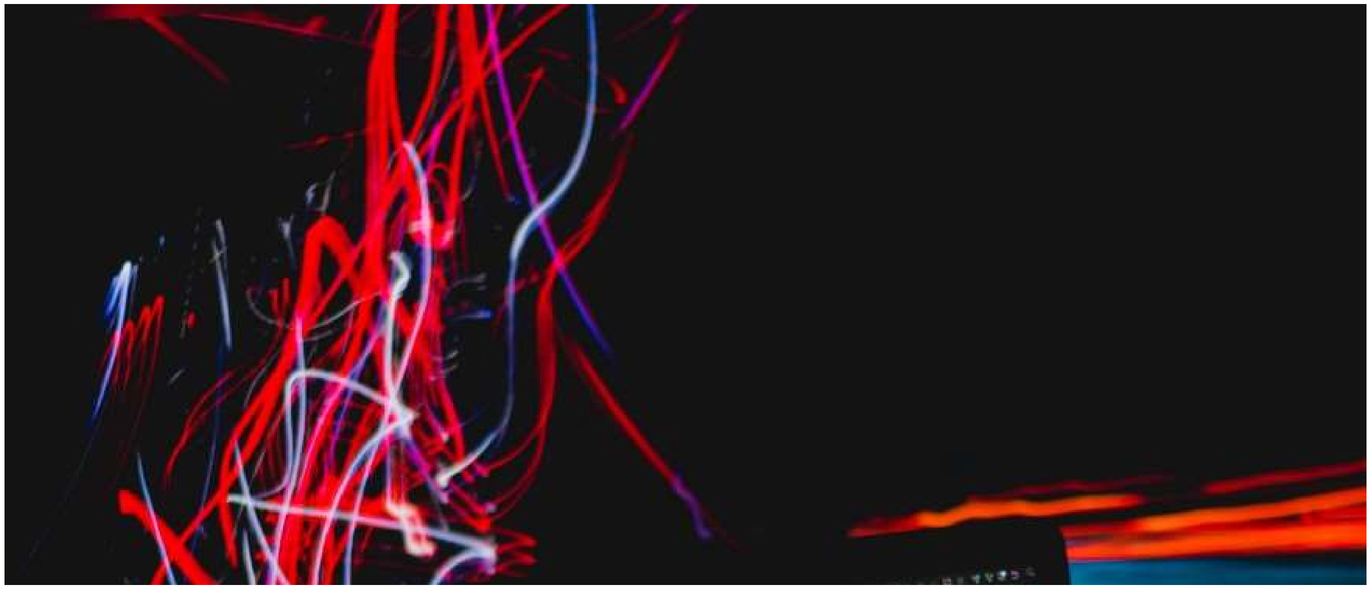
Dlib and MTCNN are both pip installable, whereas Haar Cascades and DNN face detectors require OpenCV.

```
pip install opencv-python
pip install dlib
pip install mtcnn
```

If you are using Anaconda then install these using conda commands:

```
conda install -c conda-forge opencv
conda install -c menpo dlib
conda install -c conda-forge mtcnn
```

Coding the models



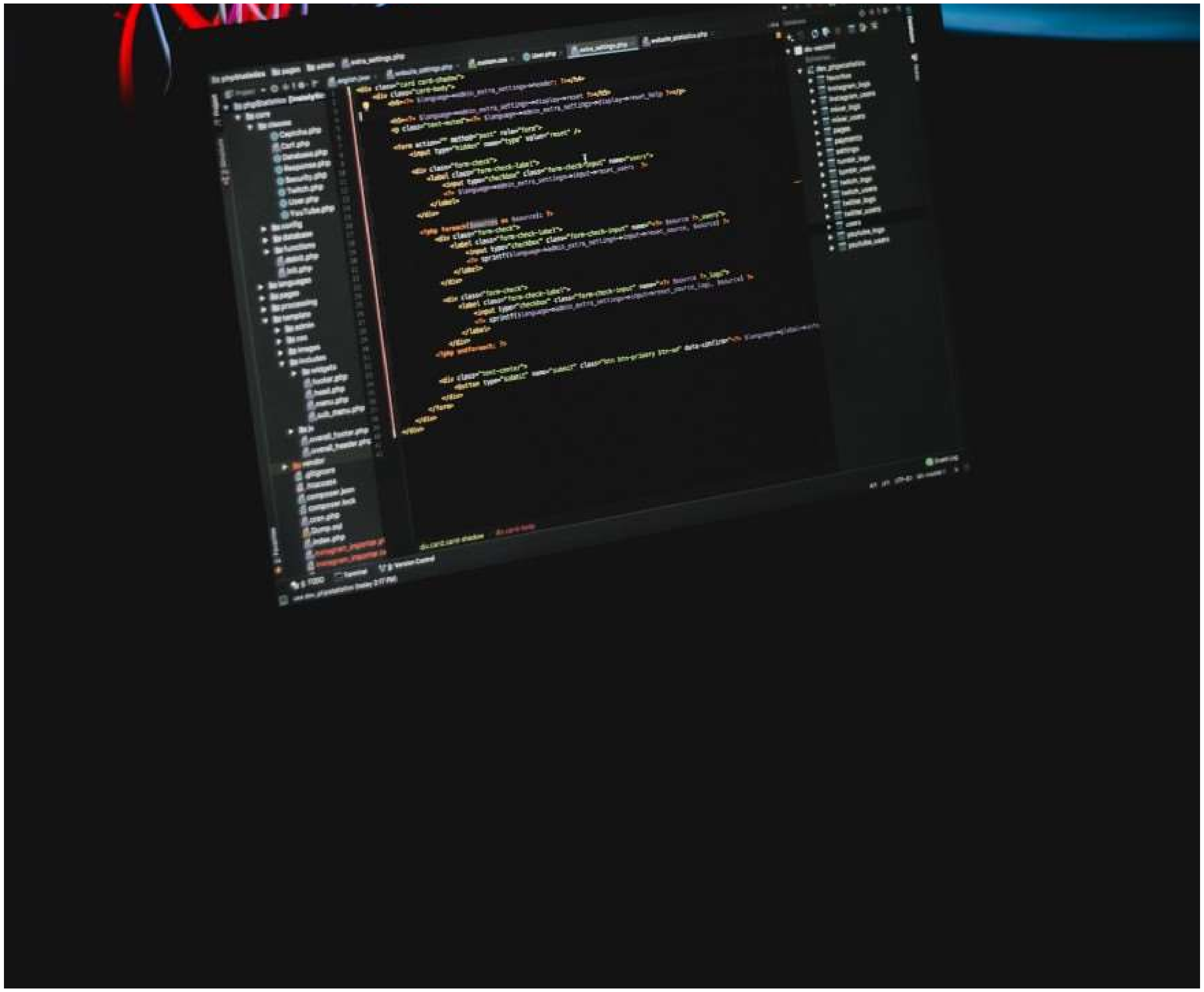


Photo by Fabian Grohs on Unsplash

In this section, I will explain the code of all these different models. The weights of HOG based descriptor of Dlib and MTCNN already come pre-bundled with their installation. The Haar cascade XML files along with the weights and layer file of the face detector of the DNN module can be downloaded from my Github repo.

Haar Cascade

```
import cv2

classifier =
cv2.CascadeClassifier('models/haarcascade_frontalface2.xml')
img = cv2.imread('test.jpg')
faces = classifier.detectMultiScale(img)# result
#to draw faces on image
for result in faces:
    x, y, w, h = result
    x1, y1 = x + w, y + h
    cv2.rectangle(img, (x, y), (x1, y1), (0, 0, 255), 2)
```

As you can see it is very easy to make predictions using Haar cascades. Just initialize the model using `cv2.CascadeClassifier` followed by detection using `cv2.detectMultiScle`. Then loop over all the faces and draw them on to the image.

Dlib HOG based frontal face detector

```
import dlib
import cv2

detector = dlib.get_frontal_face_detector()
img = cv2.imread('test.jpg')
gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
faces = detector(gray, 1) # result
#to draw faces on image
for result in faces:
    x = result.left()
    y = result.top()
    x1 = result.right()
    y1 = result.bottom()
    cv2.rectangle(img, (x, y), (x1, y1), (0, 0, 255), 2)
```

Dlib, unlike all the other models, works on grayscale images. It returns a ‘rectangle object of dlib module’ which not only contains the coordinates but also other information like area and center.

MTCNN

```
import cv2
from mtcnn.mtcnn import MTCNN

detector = MTCNN()
img = cv2.imread('test.jpg')
faces = detector.detect_faces(img)# result
#to draw faces on image
for result in faces:
    x, y, w, h = result['box']
    x1, y1 = x + w, y + h
    cv2.rectangle(img, (x, y), (x1, y1), (0, 0, 255), 2)
```

Load the MTCNN module from `mtcnn.mtcnn` and initialize it. The function `detect_faces` is used to find the results. This returns a JSON style dictionary which has the coordinates of the faces along with their confidence of prediction and the coordinates of facial landmarks detected.

Frontal face detector of DNN module

```
import cv2
import numpy as np

modelFile = "models/res10_300x300_ssd_iter_140000.caffemodel"
configFile = "models/deploy.prototxt.txt"
net = cv2.dnn.readNetFromCaffe(configFile, modelFile)
img = cv2.imread('test.jpg')
h, w = img.shape[:2]
blob = cv2.dnn.blobFromImage(cv2.resize(img, (300, 300)), 1.0,
(300, 300), (104.0, 117.0, 123.0))
net.setInput(blob)
faces = net.forward()
#to draw faces on image
for i in range(faces.shape[2]):
        confidence = faces[0, 0, i, 2]
        if confidence > 0.5:
            box = faces[0, 0, i, 3:7] * np.array([w, h, w, h])
        (x, y, x1, y1) = box.astype("int")
        cv2.rectangle(img, (x, y), (x1, y1), (0, 0, 255), 2)
```

Load the network using `cv2.dnn.readNetFromCaffe` and pass the model's layers and weights as its arguments. This is found on the Github page of OpenCV DNN.

> *To achieve the best accuracy run the model on BGR images resized to* `300x300` *applying mean subtraction of values* `(104, 177, 123)` *for each blue, green and red channels correspondingly.*

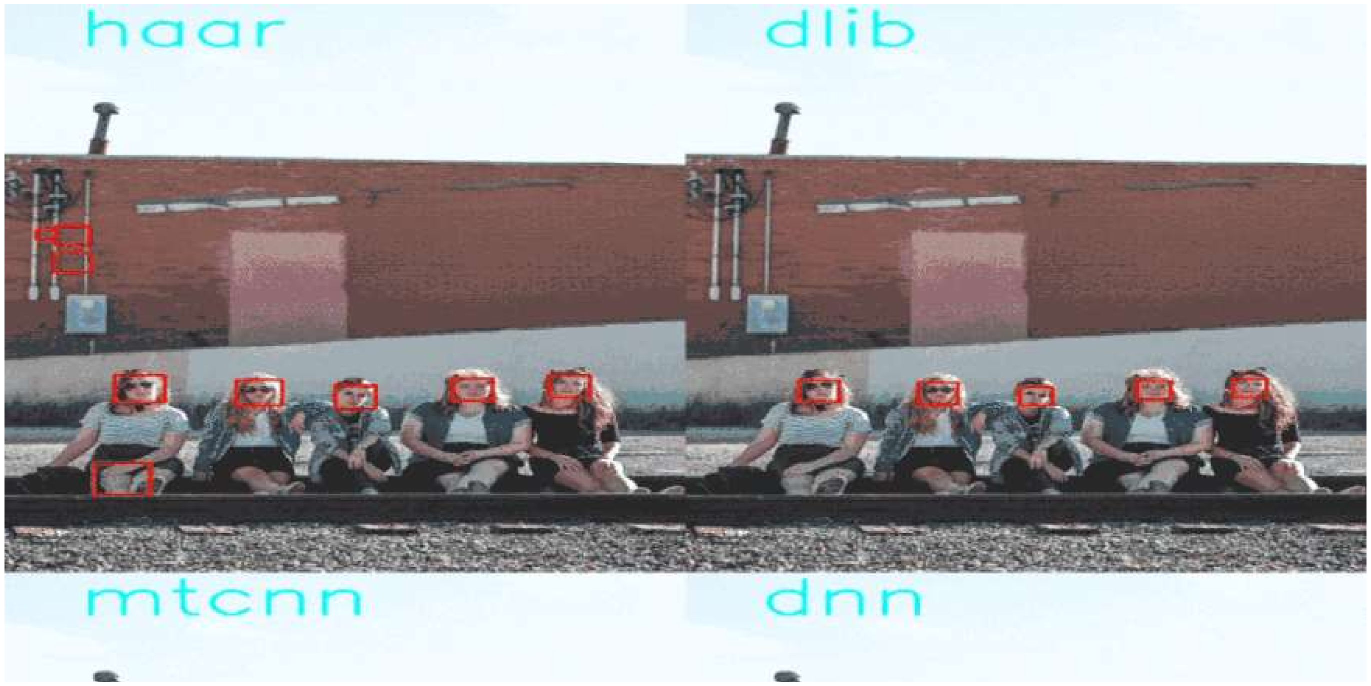
There is a lot of discrepancy around the value for green. In articles of pyimagesearch and learnopencv, I found both of them had used 117 instead of 177 so I used 117 in the `cv2.dnn.blobFromImage` function. Finally, a 4-D array is returned which contains the confidence and coordinates scaled down to the range of 0 to 1 such that by multiplying them by the original width and height to predictions for the original image can be obtained as opposed to of the 300x300 on which the model predicted.

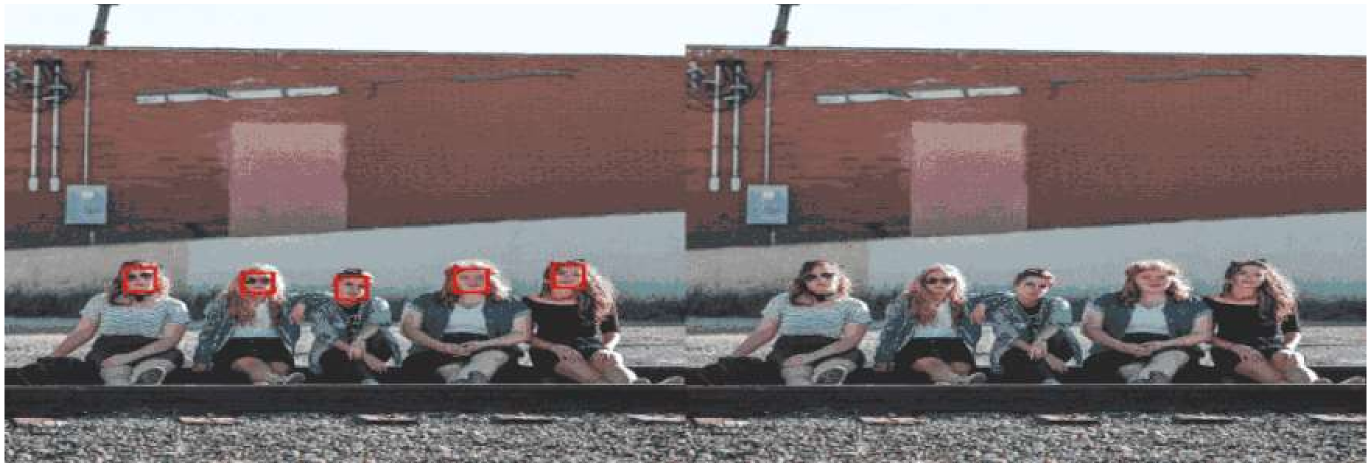
Comparing Results on Images

I created two small databases containing 10 images each with one created from photos from Unsplash and the other from Google to see how the techniques fare on both large and small images. Let's check them out one by one.

Unsplash

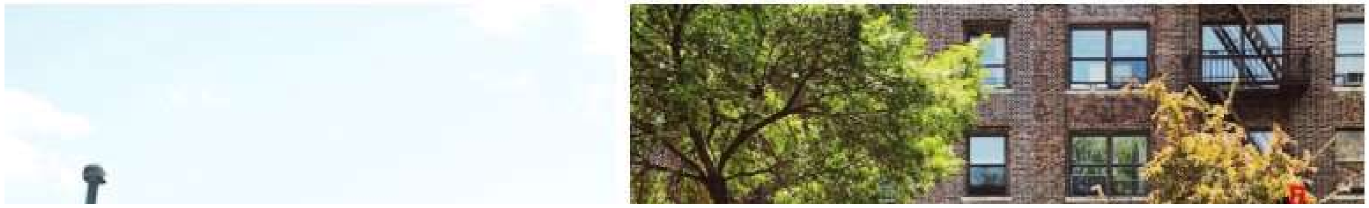
Before I start, I would like to give picture credits to Bruce Dixon, Chris Curry, Chris Murray, Ethan Johnson, Jerry Zhang, Jessica Wilson, Roland Samuel, and Tim Mossholder whose images I have used. As the average dimensions of the images were around 5000x5000, both the height and width have been reduced by half before processing.

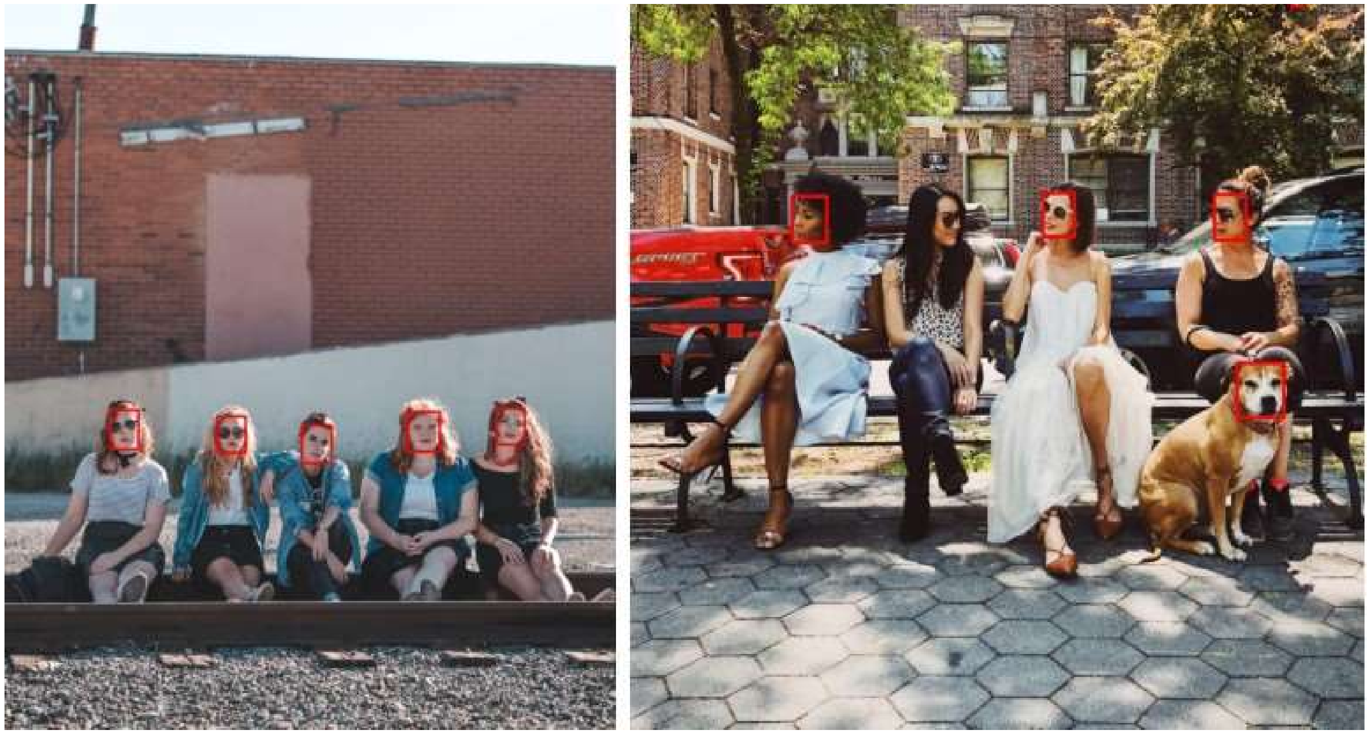




Results

Haar cascades as expected performed the worst out of all of them having a lot of false positives as well. Dlib and MTCNN had pretty even performance with one edging the others and visa-versa. DNN module had an all or nothing type of performance. On closer inspection, we can see that it did not perform well on images having small face sizes that might have occurred due to resizing it to 300x300 before starting so let's see how it would have performed if the original size was taken. To do this just change `(300, 300)` in `cv2.dnn.blobFromImage()` to original width and height respectively and remove the resize function.





Results with full Image

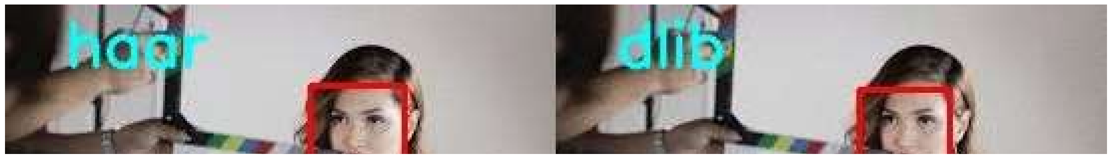
So the results have considerably improved by taking full-size images, however, this way the DNN module has not been able to make any prediction where the face size is large. Example:

No Face Detected

Google Images

These images from Unsplash were very large so I decided to check out some small images as well to see the performance on them. All the images taken had a reusability license with modification. The average dimensions are 220x220 and they are processed as it is except for the DNN module where the images are resized to 300x300 and the results were not good if original size images were used.

This created a problem for Dlib facial detector as it cannot detect faces that are less than the size of 80x80 and as the images were very small the faces were even smaller. So the image was scaled up by the factor of 2 for testing but this is a huge problem in using Dlib as the face size cannot be very small and upsampling the image will result in more processing time.



Results

Again as expected Haar performed the worst. MTCNN gave perfect results barring a couple of images and could not identify 2 faces in all. DNN came a close second and was not able to identify 3 faces. Dlib missed a lot more faces and we should also not forget that it had images upscaled by a factor of 2 already.

So before moving on to the video part let's recap the things we have learned from the results of this section.

- Haar is pretty outdated and gives the worst results generally.

- The face detection model of the DNN module of OpenCV works well but if the size of the image is very large then it can cause problems. Generally, we don't work with such 3000x3000 images so it should not be a problem.
- Dlib does not detect faces smaller than 80x80 so if working with small images make sure that you upscale them but this will increase the processing time.
- So considering the above two points, MTCNN would be the best bet if we were to deal with extreme face sizes and can be said to be leading the competition till now.

Note: Dlib's prediction sometimes misses the chin or the forehead due to the face that is was manually annotated by Davis King, the author of Dlib, so if the task you are working on cannot afford this don't use Dlib.

Comparing the models on videos

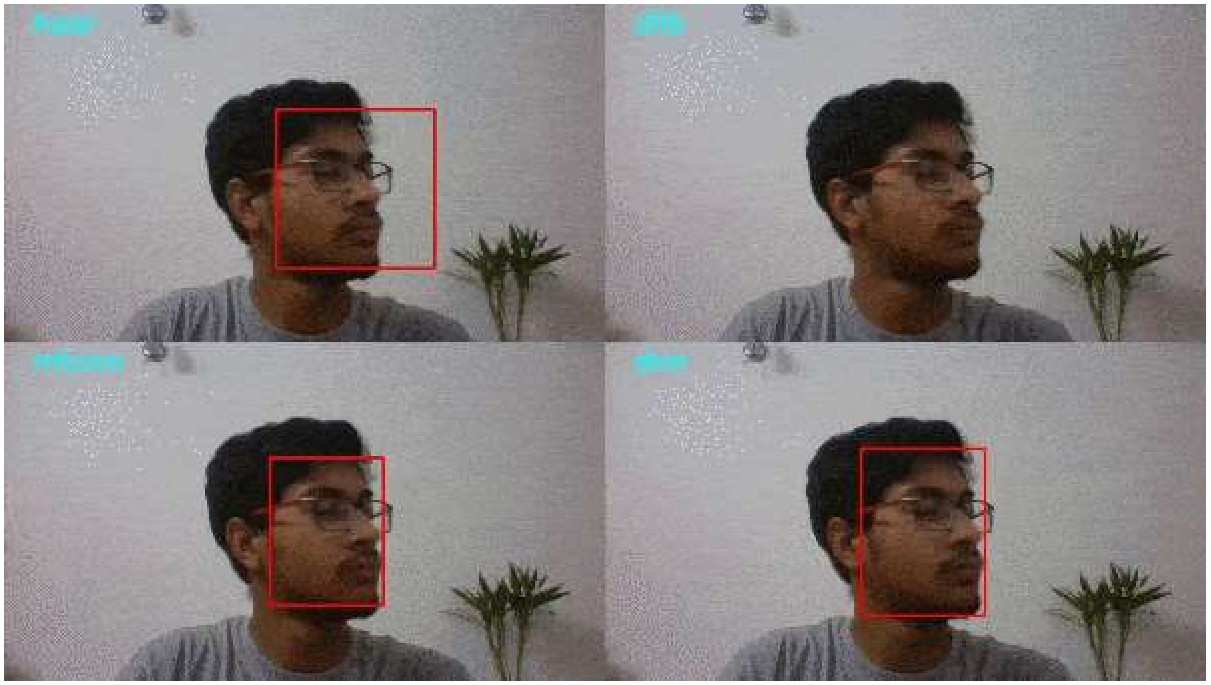
Before beginning why not clarify what objectives we will be testing our models on:

- Different angles of the face

- Head moving
- Occlusion of face
- Different lighting conditions
- Frame rate achieved

The size of each frame passed to the models is 640x360 and they are processed as it is except for the DNN model which is reduced to 300x300.

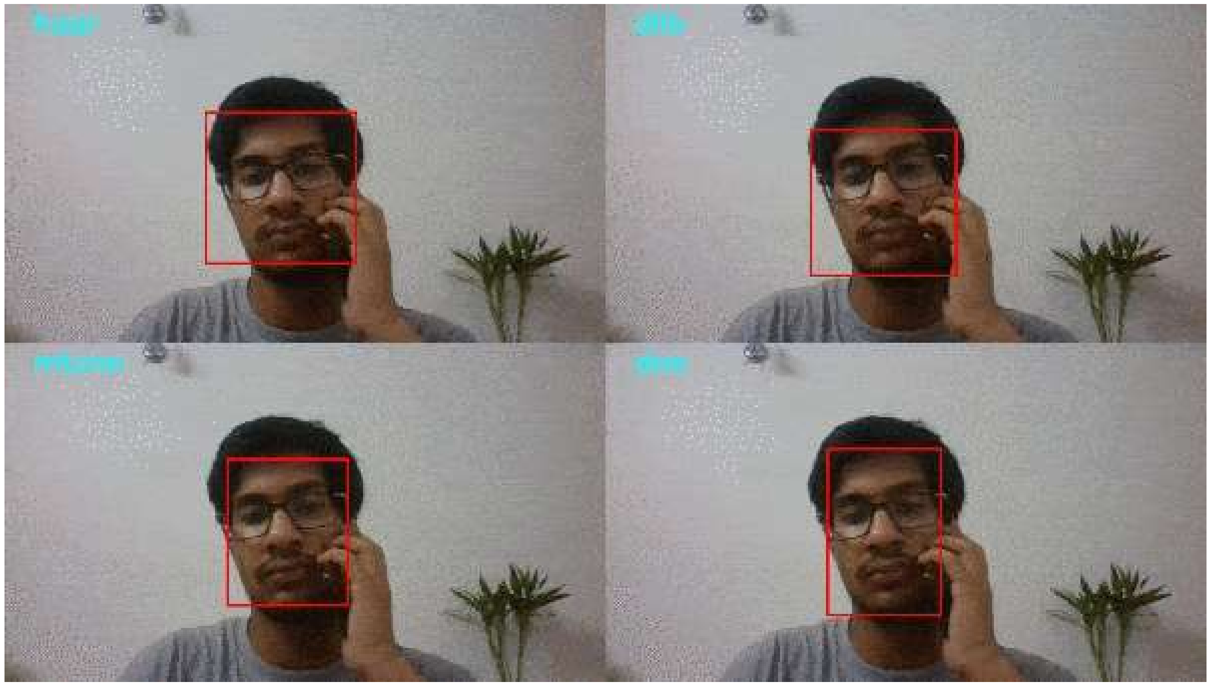
Different angles of the face and head movement



Results (compressed to reduce size)

OpenCV's DNN module hit a home run here. It was able to detect side faces up to insane angles and was largely unfazed by quick head movement as well. The others were no match for it and failed at large angles and quick movement.

Occlusion of face

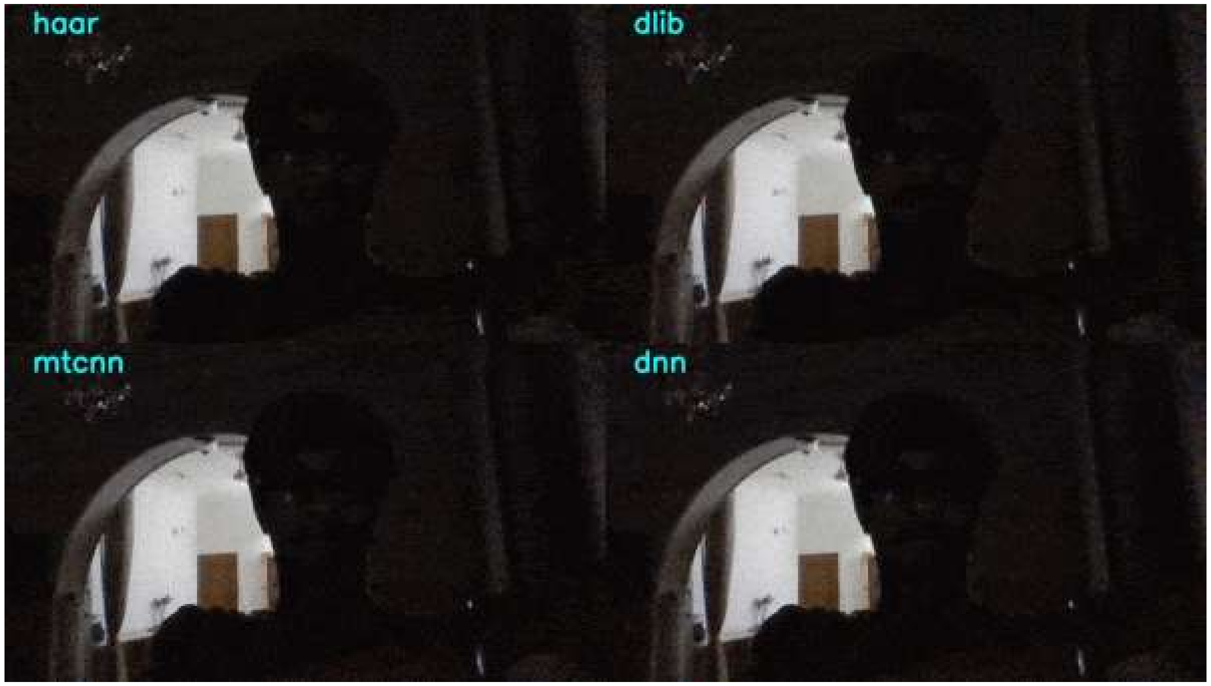


Results (Again compressed for size)

Again, the facial detection module of OpenCV's DNN is the clear winner here. Let me provide the exact results. There are a total of 642 frames in this video. DNN module was able to detect the face in 601 of them! In comparison, the second place was taken by Haar, yes Haar, which got the face in 479 of them followed by a close third in MTCNN with 464 frames. Dlib was way behind with the face detected in only 401 frames.

Different lighting conditions

The goal here was to see how well these models perform in very low light and when a light source is right behind the person.



Results

Let’s give credit where it’s due as only Haar cascade was the only model able to detect the face in the dark in a couple of frames, while the DNN model provided a false positive during that time. When the light was switched on the DNN module was the back at its work providing completely accurate predictions. Dlib’s output was a little shaky but better than Haar cascade which was able to predict even fewer frames and gave some false positives as well. The real surprise was MTCNN. It failed to detect the face in even a single frame suggesting lighting conditions need to good if it is to be used.

Frame Rate

The values reported are obtained using an Intel i5 7th gen processor and the image size passed is 640x360 except for the DNN module which is passed a 300x300 image as it has been done until now.

Haar — 9.25 fps

Dlib — 5.41 fps

MTCNN — 7.92 fps

DNN module of OpenCV — 12.95 fps

Conclusion

- The Haar Cascade classifier gave the worst results in a majority of the test along with a lot of false positives.
- Dlib and MTCNN had very similar results with a slight edge to MTCNN, but Dlib can't identify very small faces. Also if the size of images is very extreme and there is a surety that lighting will be good along with minimum occlusion and mainly front-facing faces MTCNN might give the best results as seen when we were comparing the images.
- For general computer vision problems, OpenCV's Caffe model of the DNN module is the best. It works well with occlusion, quick head movements, and can identify side faces as well. Moreover, it also gave the quickest fps among all.

All the codes and files and be found here on my Github repo. This aim was to find the face detection model best suited for face detection in an online proctoring scenario. If you are interested in knowing more about it you can read this article.

Automating Online Proctoring Using AI

Semi-automate proctoring based on vision and audio based capabilities to prevent cheating in online exams and monitor…

towardsdatascience.com

Artificial Intelligence Computer Vision Deep Learning Machine Learning Data Science