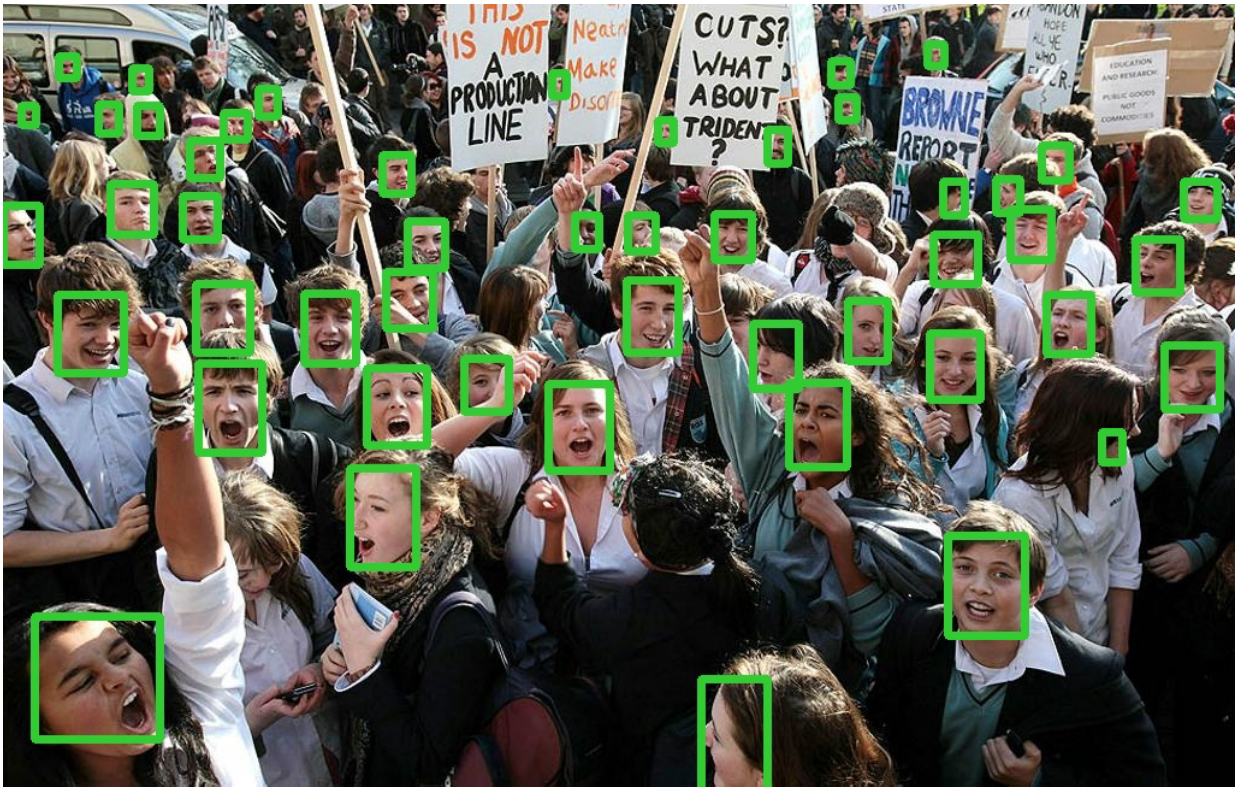
CAP 6655 – Project 1

Face detection models comparison

Matteo Bassani



Introduction

Face detection is an AI-based computer technology that can identify and locate the presence of human faces in digital photos and videos. It can be regarded as a special case of object-class detection, where the task is to find the locations and specify the sizes of all the objects that belong to a given class - in this case, faces - within a specific image or images. Face detection technology can be applied to various fields - including security, biometrics, law enforcement, entertainment and personal safety - to provide surveillance and tracking of people in real time. Face detection models often represents the basis for other computer vision approaches, such as face recognition, age estimation or face expression prediction.

In this project, my goal is to draw bounding boxes on faces using pre-trained models like Haar cascades, MTCNN, cvlib and a Caffe model using OpenCV's DNN module. Then I will compare them to find out which works the best for real-time applications.

Description

In this project, my goal is to draw bounding boxes on faces using pre-trained models like Haar cascades, MTCNN, cvlib and a Caffe model using OpenCV's DNN module. Then I will compare them to find out which works the best for real-time applications.

These approaches are tested on 27 pictures, where the first 26 are taken from the WIDER Validation Dataset (http://shuoyang1213.me/WIDERFACE/). Each picture represents a different real-world situation, for example:

- 0.jpg: Parade

- 1.jpg: Handshaking

... (Check the first 26 directory of the WIDER dataset)

- 27.jpg: No faces picture (does not belong to WIDER)

A short description of the models tested is following.

- **Haar Cascade**
  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. 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. It's important to notice that the model is better at finding frontal faces.

- **MTCNN**
  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, a fully convolutional network is used to obtain candidate windows and their bounding box regression vectors. 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.

- **cvlib**
  cvlib is a simple, easy-to-use, high level, open-source Computer Vision library for Python. The library was developed with a focus on enabling easy and fast experimentation. Most of the Guided principles that cvlib has are heavily inspired by the famous Keras library (Deep Learning library used on top of TensorFlow).

- **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.

Results

For each picture and for each model, the number of faces detected by the model are plotted in an hystogram. Then, the number of faces detected by the models are compared to the real number of faces in the picture (extracted from WIDER labels).

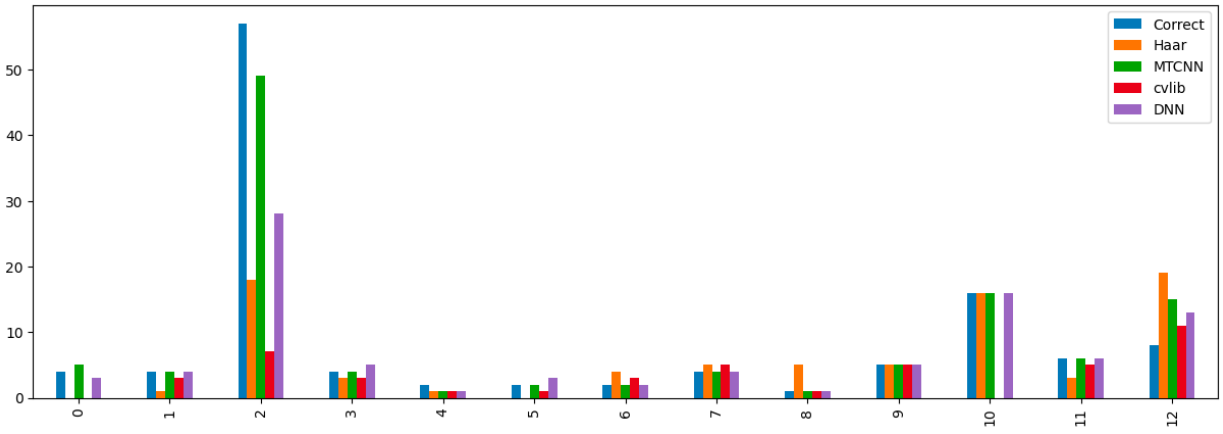


Figure 1 - Histograms for the first 13 images

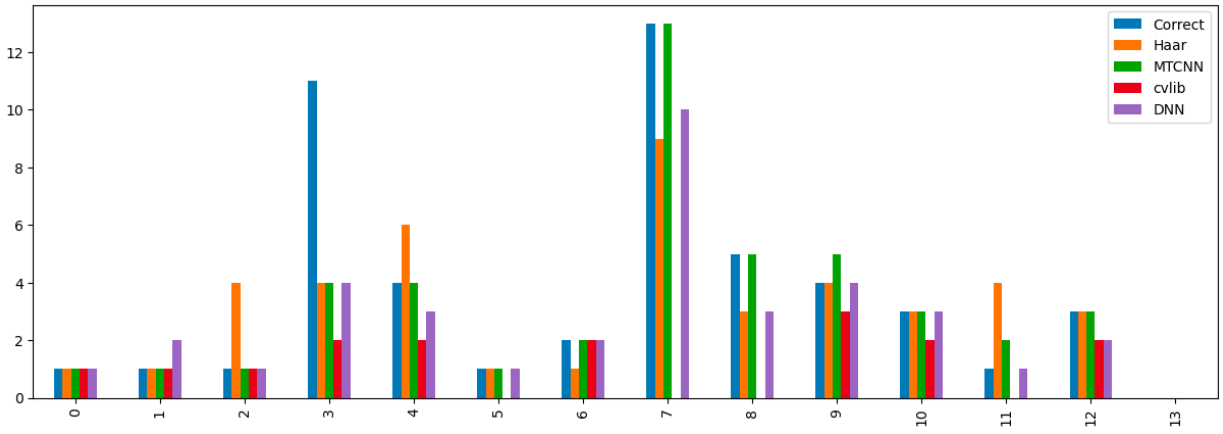
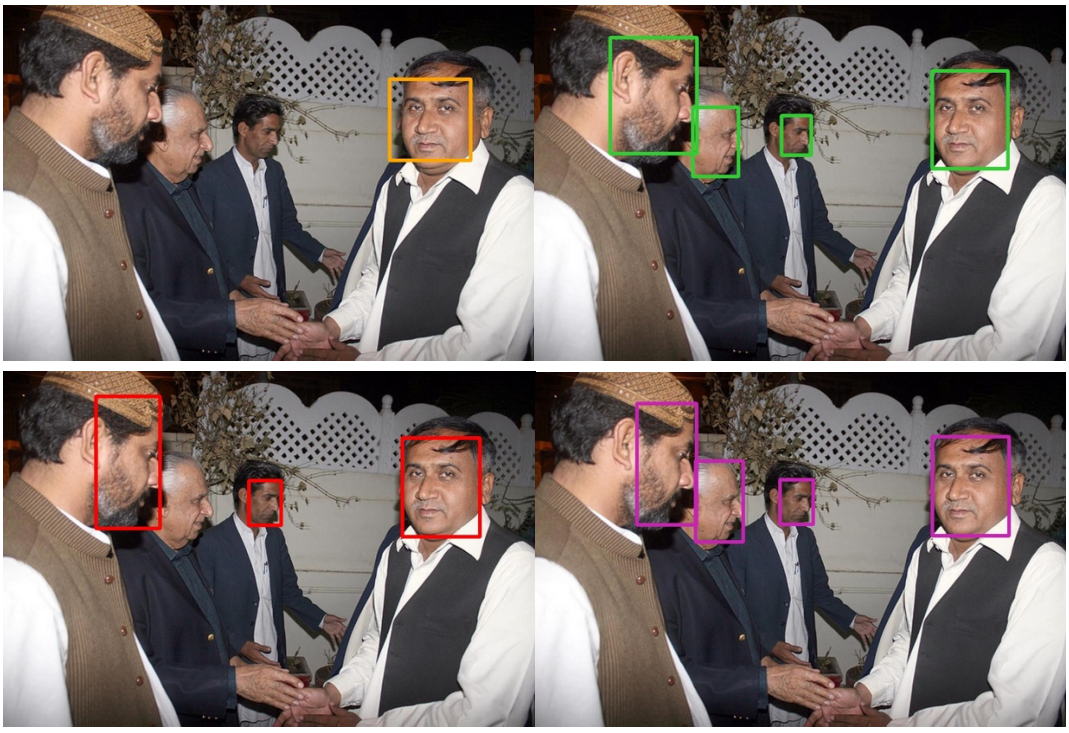


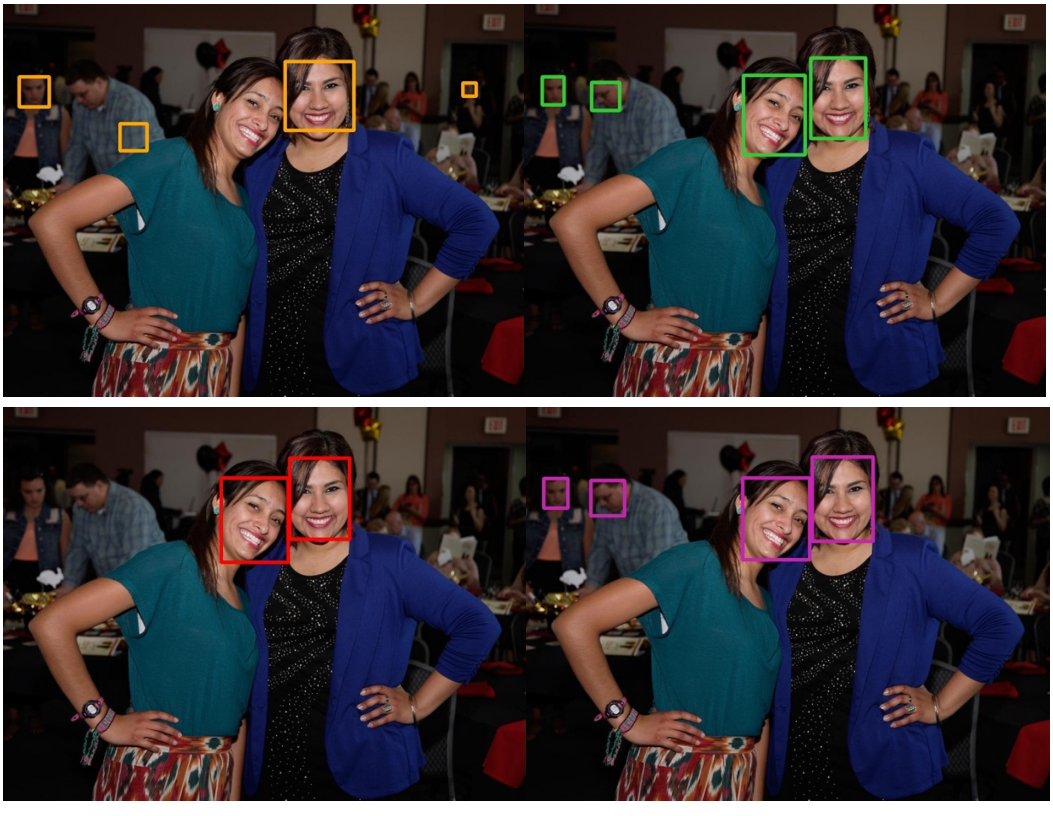
Figure 2 Histograms for the last 14 images

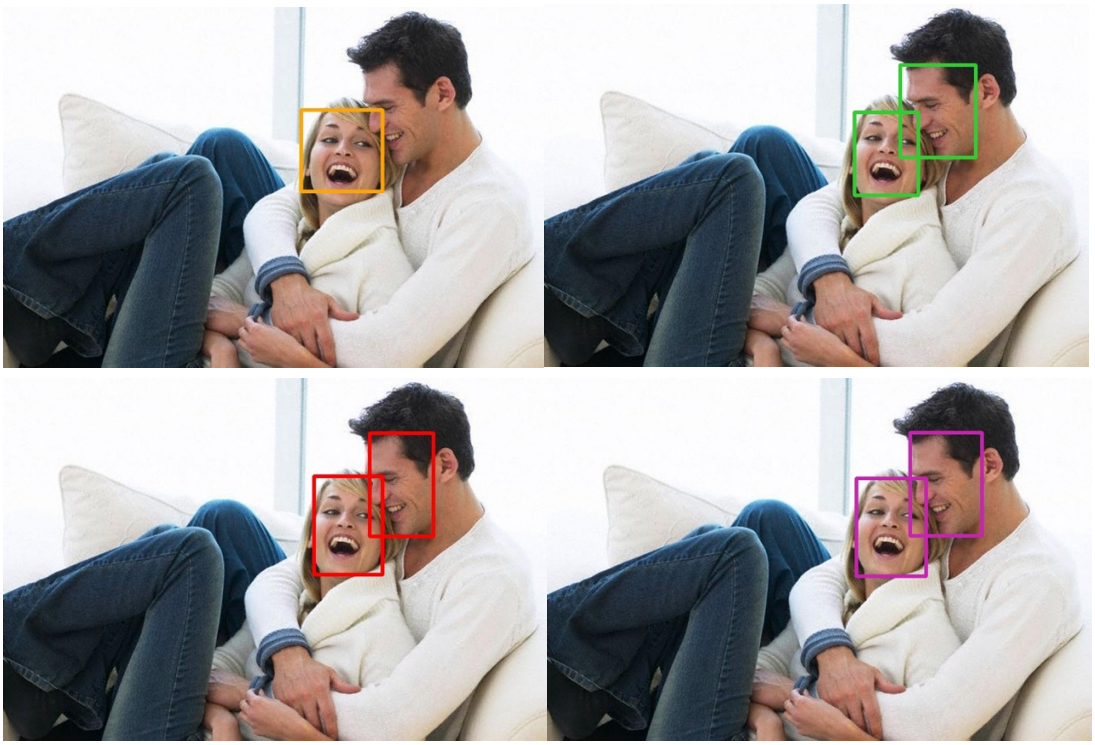
To properly compare performances of face detection methods we should consider also the position of the faces detected, and so the bounding boxes coordinates. In fact, the number of faces detected alone is not sufficient to prove that a model is good.

However, since the final goal is to use the detection model to perform further operations with the faces detected, we can also compare the models by simply observing the results obtained for different pictures.

Orange boxes : Haar - Green boxes : MTCNN - Red boxes : cvlib - Purple boxes : DNN








IS NOT A PRODUCTION LINE
CUTS? WHAT ABOUT TRIDENT?
BROWNE REPORT

Now, we measure a sort of accuracy for each approach. This measure is obtained by summing the total number of faces detected by each model and divided by the real number of faces in every picture. Obviously, we're not considering that there might be false positives that can raise this measure. However, by doing this we can have a clue on which method can localize the highest number of faces.

--- Accuracies ---

- Haar: 0.7515151515151515
- MTCNN: 0.9636363636363636
- cvlib: 0.3696969696969697
- DNN: 0.7757575757575758

It's interesting also to measure the time required by each method to perform the computation of all the pictures:

--- Time ---

- Haar: 7.19 s
- MTCNN: 34.7 s
- cvlib: 2.24 s
- DNN: 7.24 s

Next, we will compare the models on videos. In particular, we will be testing the models on:

- Different angles of the face
- Head moving
- Occlusion of face
- Different lighting conditions

Figure 3 This is actually a video, try clicking it

Analysis

As we can see Haar cascades performed bad, having a lot of false positives as well. This happens both in images and in vides. This method is quite good at detecting frontal faces, but not so good at detecting faces from different angles. Especially in videos, the tracking is bad with many false positives, which represents a great problem if we want to perform other tasks with the faces detected. An advantage of this approach is that it doesn't require many computational resources, and the detection is fast both for images and videos.

MTCNN instead had very good performance, with a great accuracy in detecting faces in images. It's the method which detected the greatest number of faces in the sample pictures. It performed very well also in video tracking, with a great accuracy in almost every situation. This algorithm is also able to track the position of eyes, nose and mouth, so it can be useful in some applications. However, its greatest drawback consists in the expensiveness of this approach. In fact, it took five times Haar's time to perform detection in the same pictures.

cvlib instead performed very bad in images. In some pictures with many faces it couldn't detect most of them, while in pictures with a low number of faces it performed better. In video tracking instead it performed good, by following the face even when in different orientations. A great advantage is that this method is very optimized: it was the fastest of all both in images and videos.

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. In images it got similar performances to MTCNN, however in pictures with many faces, some of them are not detected. In videos, It was able to detect side faces up to different angles and was largely unfazed by quick head movement as well. Also, the algorithm is pretty fast, it took more or less Haar's time to perform face detection in 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's also less expensive than MTCNN in computational resources.

Flow Chart

Flow chart for face detection is pretty straightforward.

Here, only Haar cascade classifier flow chart is reported, but it's very simple to represent a similar flow chart also for the other three methods.

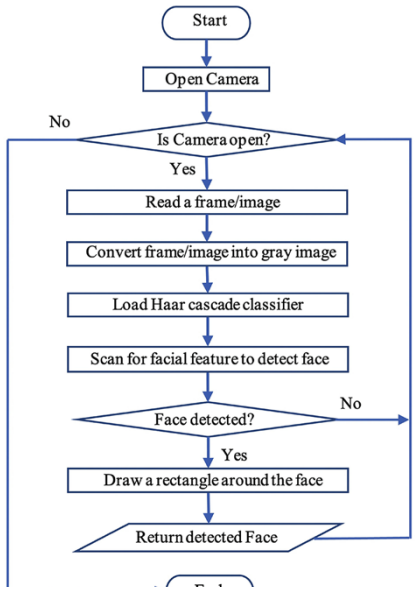


Figure 4 Haar cascade flow chart

Code

To access the code of this project:

https://github.com/Matteo299/FaceDetectionComparator.git

- face_detector_comparator.ipynb is the main notebook.
- <name_method>.ipynb is for webcam face detection.
- <name_method>test.py is for video face detection.