Object detection using Haar cascade

Outline

- Review the Viola-Jones algorithm for face recognition
- OpenCV implementation
- Extension to general objects detection
- Pros and Cons

The Viola-Jones algorithm

- Viola, Paul, and Michael Jones. "Rapid object detection using a boosted cascade of simple features." *Proceedings of the 2001 IEEE computer society conference on computer vision and pattern recognition. CVPR 2001*. Vol. 1. IEEE, 2001.
- The algorithm
  - Detects faces regardless of their scale and location in the image
  - Real-time performance
- The algorithm can be trained to detect arbitrary objects

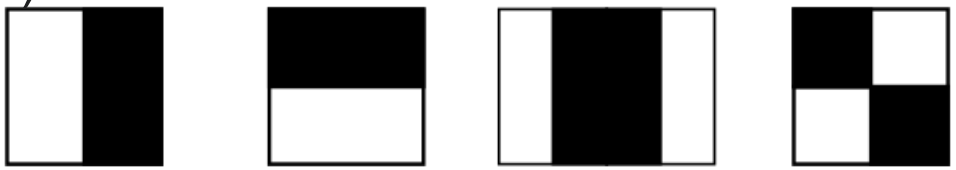
Haar Features

- The key aspect is to detect relevant features in human face (eyes, nose, lips, etc.)
- Face properties
  - The eye region is darker than the upper-cheeks.
  - The nose bridge region is brighter than the eyes.
- The composition of features
  - Location and size of eyes, moth, nose

Haar Features

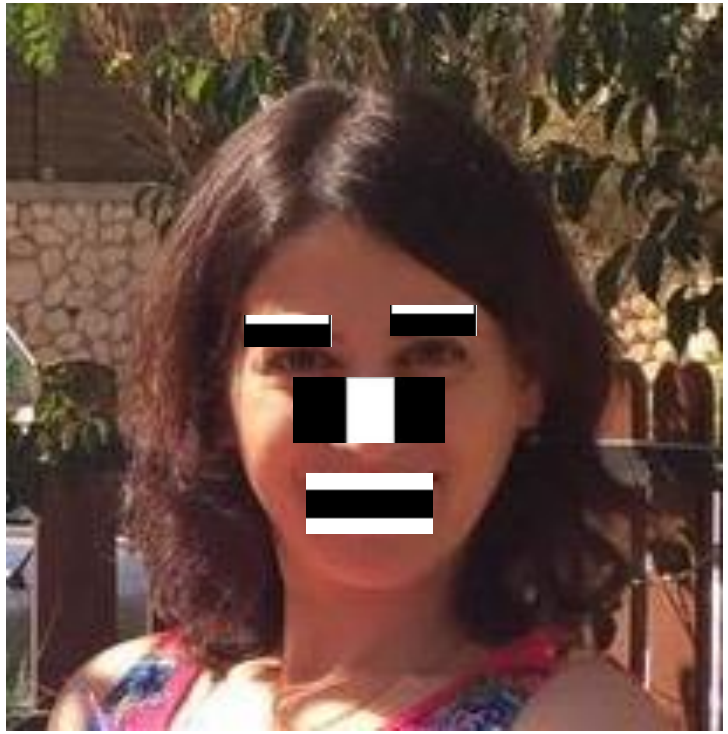
- Haar features are white and black images
- This images slide across the images from left to right, and top to bottom (like in image filtering)

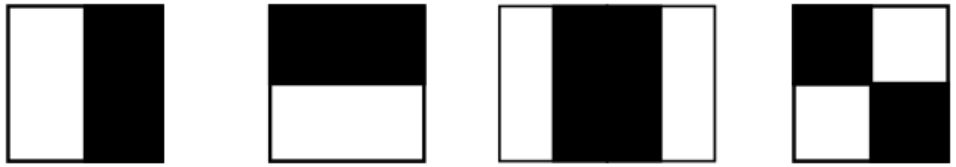
- At each location we calculate
  value = Σ (pixels in black area) - Σ (pixels in white area)



Haar Features

- Face properties
  - The eye region is darker than the upper-cheeks.
  - The nose bridge region is brighter than the eyes.
- The composition of features
  - Location and size of eyes, moth, nose
- The features are sought in the image to detect a face





Feature calculation

- Computing Haar features as each (x,y) location can be computationally expensive.
- Viola and Jones, proposed an intermediate representation for the image : the integral image.
  - $\text{Integral\_Img}(x, y) = \sum_{\substack{x' \leq x \\ y' \leq y}} I(x', y')$

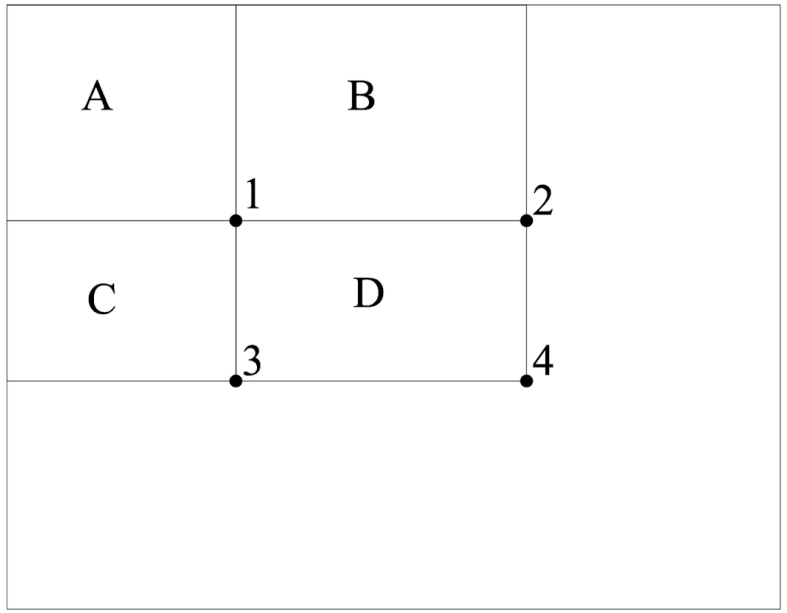
31	2	4	33	5	36
12	26	9	10	29	25
13	17	21	22	20	18
24	23	15	16	14	19
30	8	28	27	11	7
1	35	34	3	32	6

Original Image

31	33	37	70	75	11
43	71	84	127	161	222
56	101	135	200	254	333
80	148	197	278	346	444
110	186	263	371	450	550
111	222	333	444	555	666

Integral Image

Feature calculation



31	2	4	33	5	36
12	26	9	10	29	25
13	17	21	22	20	18
24	23	15	16	14	19
30	8	28	27	11	7
1	35	34	3	32	6

Original Image

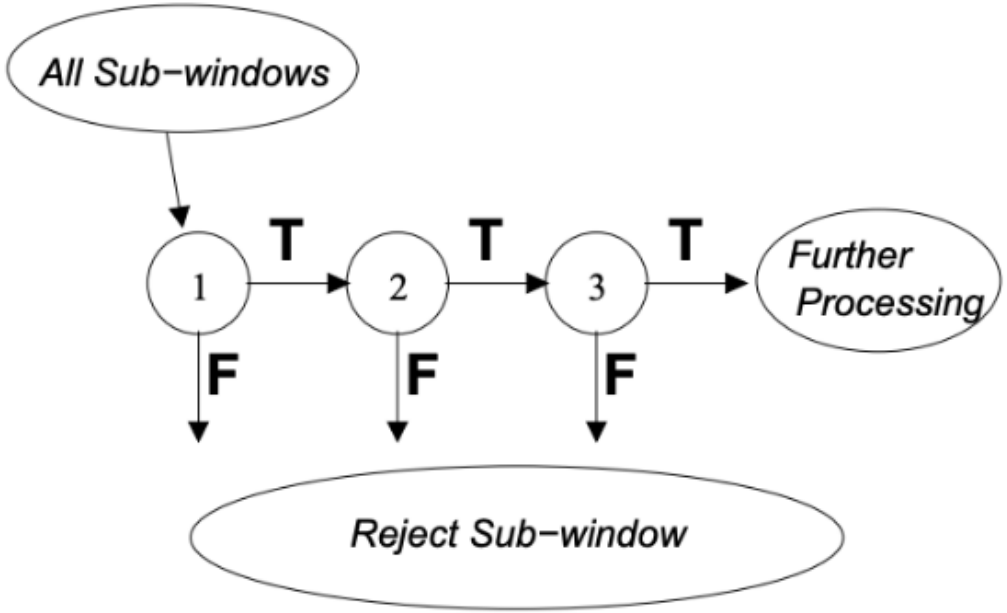
31	33	37	70	75	11
43	71	84	127	161	222
56	101	135	200	254	333
80	148	197	278	346	444
110	186	263	371	450	550
111	222	333	444	555	666

Integral Image

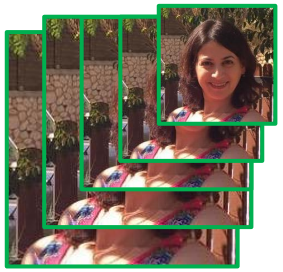
15+16+14+28+27+11=
= 450-186-254+101=111

Cascading Classifier

- The computation is still expensinve
- Cascading classifier
  - At each location the window undergo a series of tests
  - Each test is more computationally expensive then previous
  - If a test fail, the window is discarded

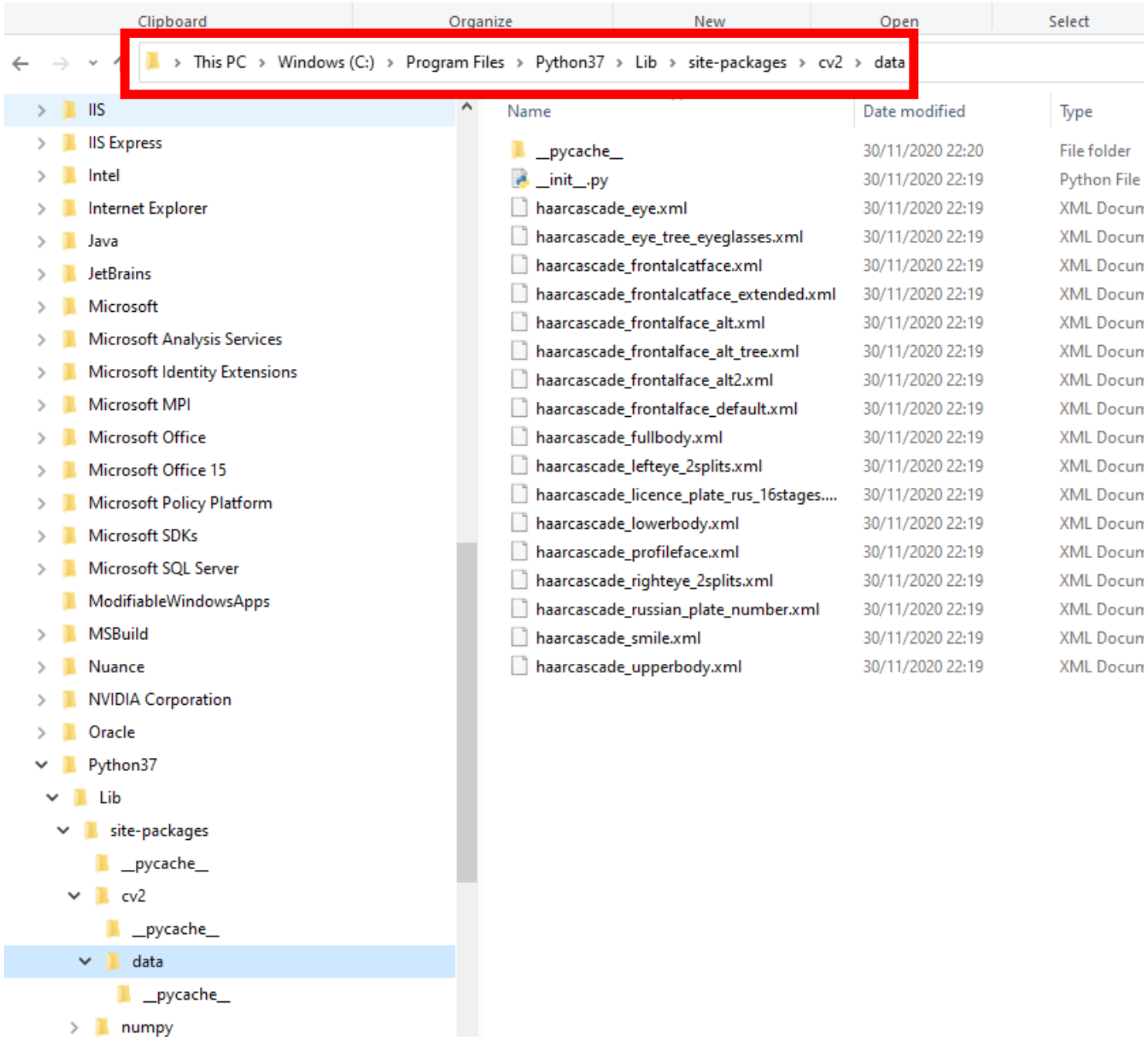


- The algorithm is applied on a pyramid of image scales



OpenCV implementation

- In OpenCV there is a pretrained model for face detection (and a number of other objects)



OpenCV implementation

```
face_cascade = cv2.CascadeClassifier(cv2.data.haarcascades +
                                     'haarcascade_frontalface_default.xml')
eye_cascade = cv2.CascadeClassifier(cv2.data.haarcascades + 'haarcascade_eye.xml’)
```

```
objects=cv2.CascadeClassifier.detectMultiScale(image[, scaleFactor[, minNeighbors[, flags[, minSize[, maxSize]]]]])
```

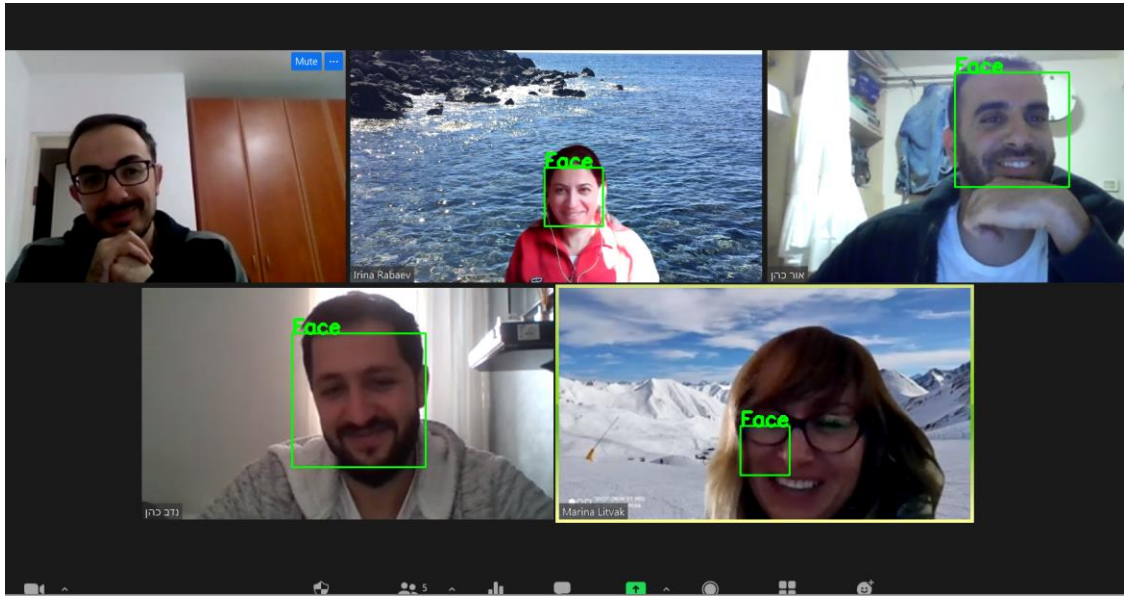
image	Matrix of the type CV_8U containing an image where objects are detected.
objects	Vector of rectangles where each rectangle contains the detected object, the rectangles may be partially outside the original image.
scaleFactor	Parameter specifying how much the image size is reduced at each image scale.
minNeighbors	Parameter specifying how many neighbors each candidate rectangle should have to retain it.
flags	Parameter with the same meaning for an old cascade as in the function cvHaarDetectObjects. It is not used for a new cascade.
minSize	Minimum possible object size. Objects smaller than that are ignored.
maxSize	Maximum possible object size. Objects larger than that are ignored. If maxSize == minSize model is evaluated on single scale.

OpenCV implementation

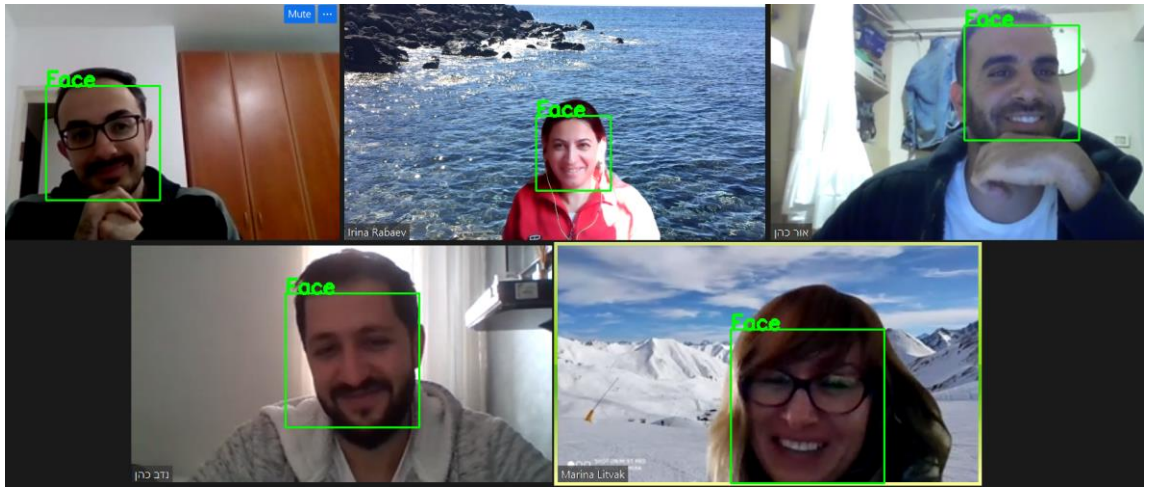
```
face_cascade = cv2.CascadeClassifier(cv2.data.haarcascades +
                                     'haarcascade_frontalface_default.xml')

img = cv2.imread(imageFilename)
gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

faces = face_cascade.detectMultiScale(gray, scaleFactor=1.1, minNeighbors=5)
for (x,y,w,h) in faces:
    cv2.rectangle(img,(x,y),(x+w,y+h),(255,0,0),2)
    cv2.putText(img, 'Face', (x, y), cv2.FONT_HERSHEY_SIMPLEX, 1, (255, 0, 0), 3)
```



faces = face_cascade.detectMultiScale(gray, scaleFactor=1.1, minNeighbors=5)



faces = face_cascade.detectMultiScale(gray, scaleFactor=1.05, minNeighbors=7, minSize=(100,100))

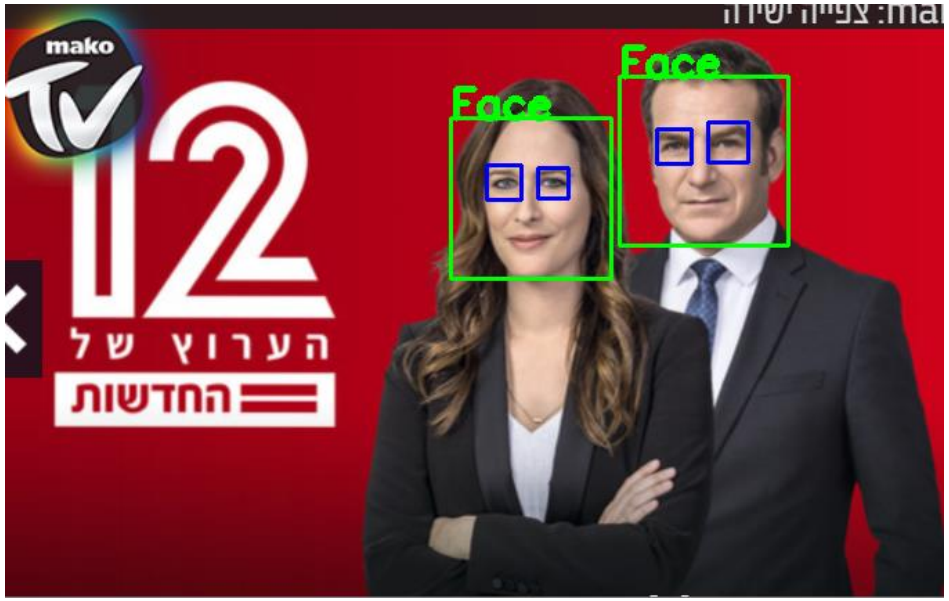
What about eyes?

```
faces = face_cascade.detectMultiScale(gray, scaleFactor=1.05, minNeighbors=7, minSize=(100,100))

eye_cascade = cv2.CascadeClassifier(cv2.data.haarcascades + 'haarcascade_eye.xml')


for (x,y,w,h) in faces:
    cv2.rectangle(img,(x,y),(x+w,y+h),(0,255,0),2)
    cv2.putText(img, 'Face', (x, y), cv2.FONT_HERSHEY_SIMPLEX, 1, (0,255, 0), 3)
    roi_gray = gray[y:y+h, x:x+w]

    eyes = eye_cascade.detectMultiScale(roi_gray)
    for (ex,ey,ew,eh) in eyes:
        cv2.rectangle(img,(ex+x,ey+y),(x+ex+ew,y+ey+eh),(255,0,0),2)
```

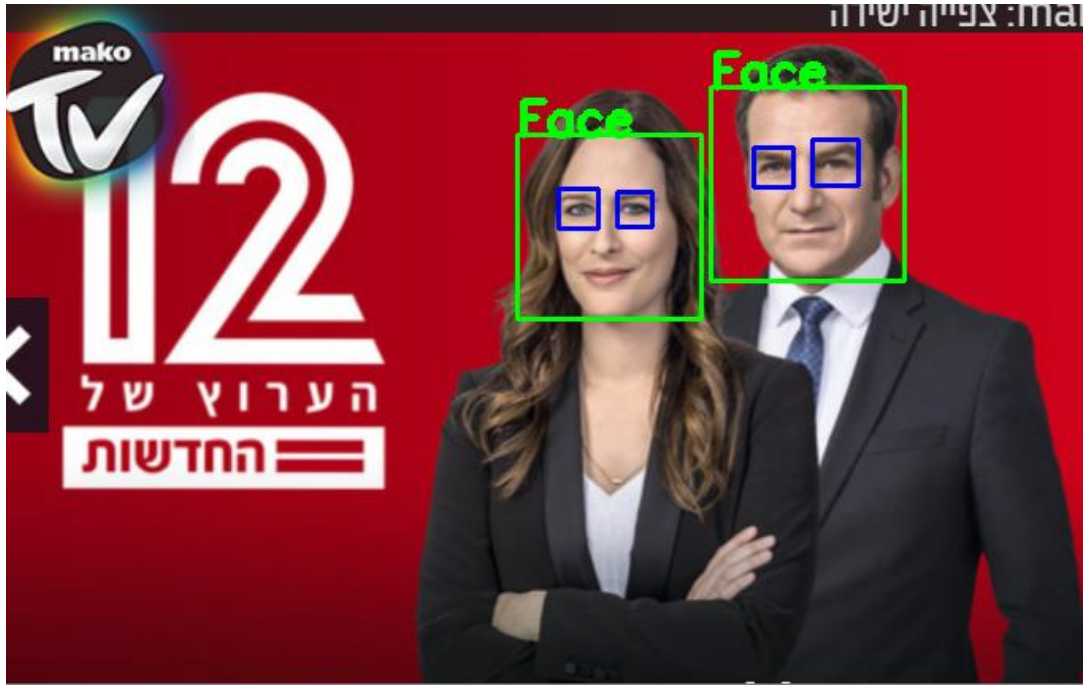


minNeigbours parameter explanation

- The detector run on multiple scales
- So will get responses from multiple scales even for a single face
- The region in classified as a face if the number of responses for this face is higher than minNeighbors.

minNeigbours parameter explanation

minNeigbours=0



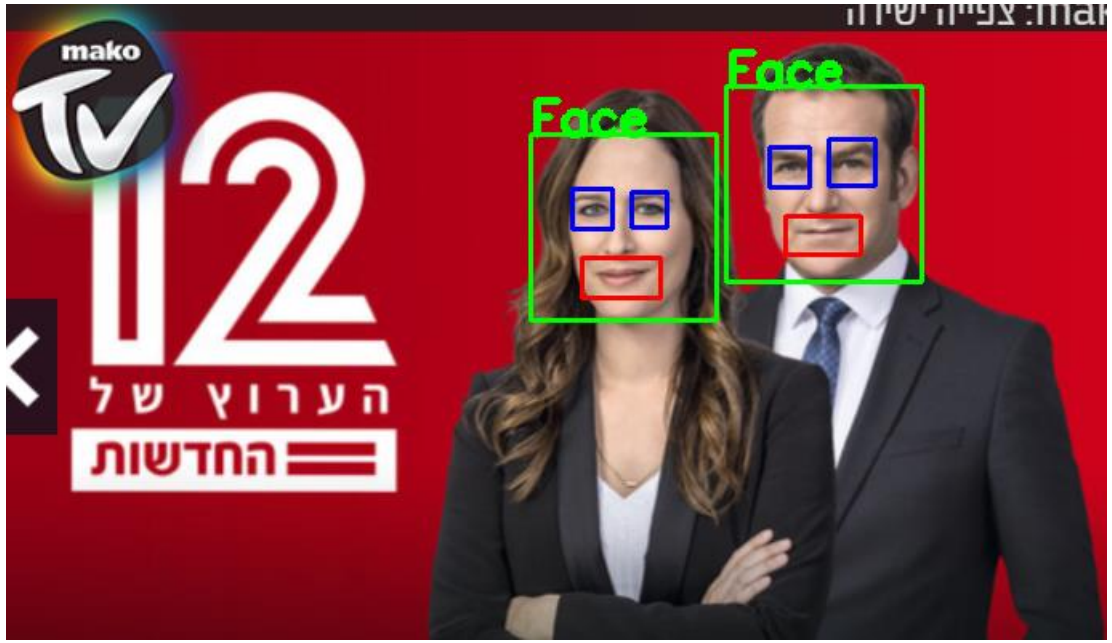
minNeigbours=5

What about smiles

```
smile_cascade = cv2.CascadeClassifier(cv2.data.haarcascades + 'haarcascade_smile.xml')

smiles = smile_cascade.detectMultiScale(roi_gray)

for (ex,ey,ew,eh) in smiles:
    cv2.rectangle(img,(ex+x,ey+y),(x+ex+ew,y+ey+eh),(0,0,255),2)
```



Viola-Jones algorithm - summary

- Pros
  - Simple
  - Real-time performance
  - Detects faces at different scales
- Cons
  - Works well only for front view faces
  - Prone to false positives
  - Does not work under occlusion or non-frontal faces
  - Tedious to tune the parameters

For highly accurate object detectors use deep-learning methods