We found out, that ‚scaleFactor‘ and ‚minNeighbors‘ are important parameters used in the ‚detectMultiScale‘ method in OpenCV. These parameters are crucial for effectively detecting objects like faces, eyes, or smiles in images. At the beginning we got poor results, after tuning these parameters the performance improved.   
  
Let's first understand the purpose and impact of the parameters:

**scaleFactor:**

The parameter specifies how much the image size is reduced at each image scale. This window is scaled down in size for each subsequent pass, allowing the algorithm to detect objects at various sizes (since an object can appear larger or smaller depending on its distance from the camera).

A smaller scaleFactor increases the chance of detecting smaller objects but also increases the computation time because more scales need to be processed. A larger scaleFactor reduces computation time but might miss smaller objects.

Typical Values are: 1.1, 1.2, 1.3 etc.   
A scale factor of 1.1 means that in each subsequent round, the search window is scaled down to 90% of its previous size.

**minNeighbors:**

The minNeighbors parameter specifies the number of neighbors a rectangle should have to be retained as detection. After the size of the moving window is set by the scaleFactor, the algorithm needs to decide whether a given window location contains the object (like a face or an eye). minNeighbors sets the condition for reliability of detection.

A higher value results in fewer detections but with higher quality. A lower value increases the number of detections but also increases the number of false positives.

Typical Values are small integer values, like 3, 4, 5.

**minSize**:

The minSize parameter sets the minimum size of the detection window. Any object smaller than this size will not be considered during the detection process.

In images where there is a lot of noise or smaller, irrelevant features, setting an appropriate minSize can help to reduce false positives.  
  
Typical Values are [20, 20] or [30, 30]

Especially we had to find the right balance for scaleFactor and minNeighbors:

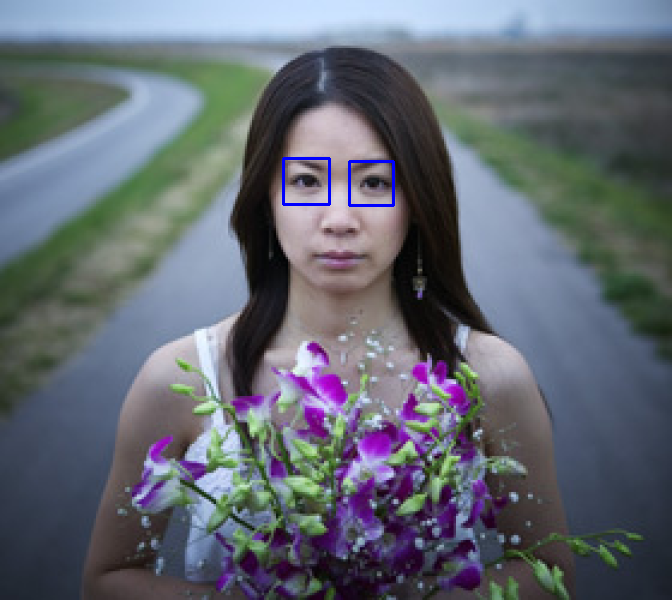
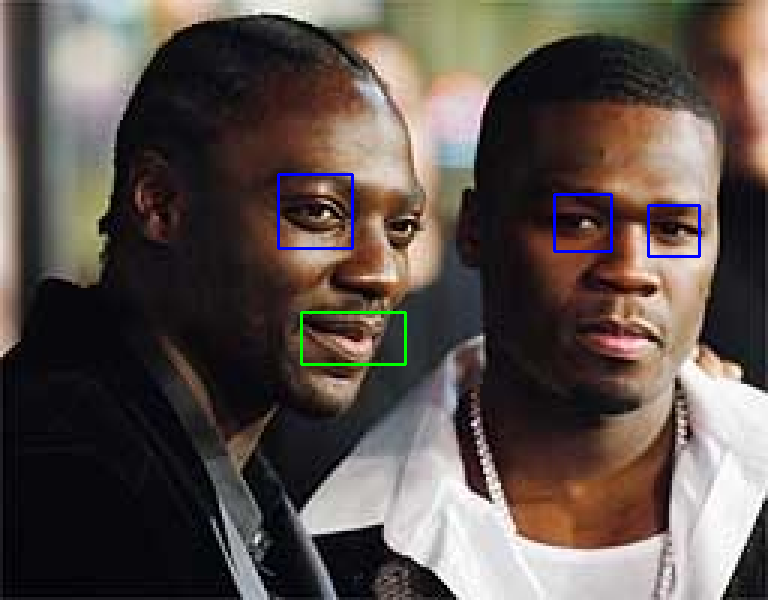
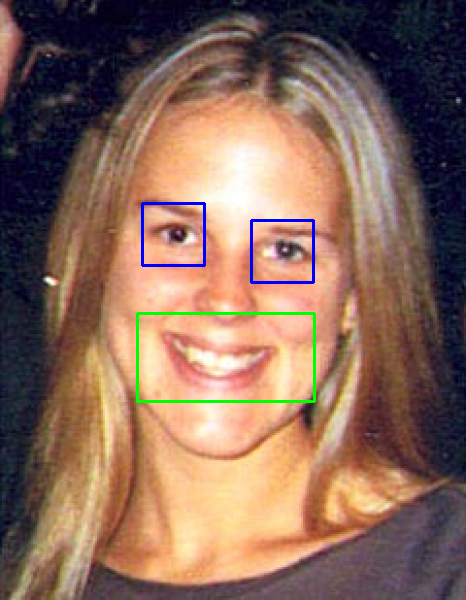
* If we are missing objects, we have to decrease scaleFactor or minNeighbors
* If we are getting to many false positives, we have to increase minNeighbors or scaleFactor
* Smaller scaleFactor uses more computational power, but that is not a problem for us, since we don’t use our application in real-time.

# Detecting eyes within the face ROI

eyes = eye\_cascade.detectMultiScale(roiGray, scaleFactor=1.3, minNeighbors=3, minSize=(20, 20))

# Detecting smiles within the face ROI

smiles = smile\_cascade.detectMultiScale(roiGray, scaleFactor=1.7, minNeighbors=20, minSize=(25, 25))

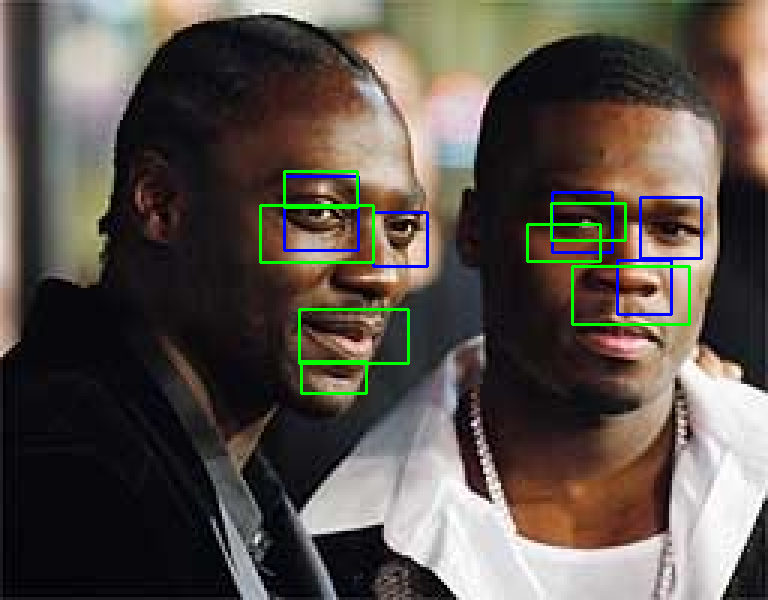
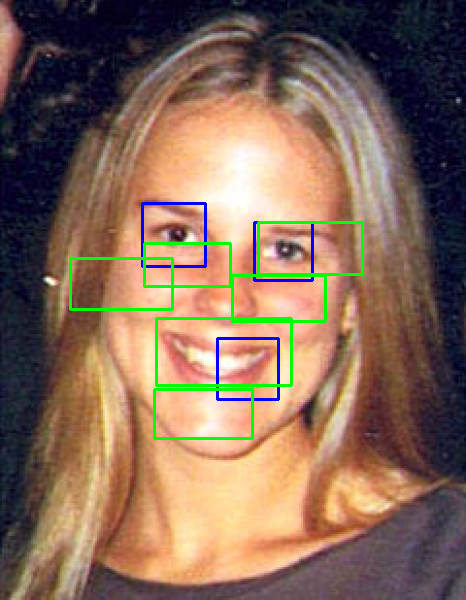


# Detecting eyes within the face ROI

eyes = eye\_cascade.detectMultiScale(roiGray, scaleFactor=1.1, minNeighbors=3, minSize=(20, 20))

# Detecting smiles within the face ROI

smiles = smile\_cascade.detectMultiScale(roiGray, scaleFactor=1.1, minNeighbors=10, minSize=(25, 25))

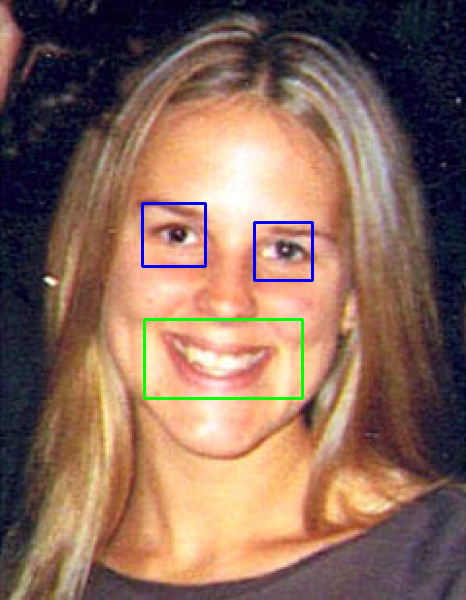


eyes within the face ROI

        eyes = eye\_cascade.detectMultiScale(roiGray, scaleFactor=1.1, minNeighbors=7, minSize=(20, 20))

        # Detecting smiles within the face ROI

        smiles = smile\_cascade.detectMultiScale(roiGray, scaleFactor=1.3 , minNeighbors=20 , minSize=(25, 25))



Faces

faces = faceCascade.detectMultiScale(gray, scaleFactor=1.2 , minNeighbors=5, minSize=(30, 30))

for (x, y, w, h) in faces:

        # Extracting the region of interest (ROI) for the face in grayscale and color images

        roiGray = gray[y:y+h, x:x+w]

        roiImg = img[y:y+h, x:x+w]

        # Drawing a rectangle around the detected face in the color image

        cv2.rectangle(img, (x, y), (x + w, y + h), (0, 0, 255), 2)

