Haar Cascade, Face Detection

May 25, 2020

0.1 Face detection using OpenCV

One older (from around 2001), but still popular scheme for face detection is a Haar cascade classifier; these classifiers in the OpenCV library and use feature-based classification cascades that learn to isolate and detect faces in an image. You can read the original paper proposing this approach here.

Let's see how face detection works on an exampe in this notebook.

In [1]:
```
# import required libraries for this section
%matplotlib inline
import numpy as np
import matplotlib.pyplot as plt
import cv2
```

In [2]:
```
# load in color image for face detection
image = cv2.imread('images/multi_faces.jpg')

# convert to RBG
image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)

plt.figure(figsize=(20,10))
plt.imshow(image)
```

Out[2]:
```
<matplotlib.image.AxesImage at 0x7f48390f9160>
```

To use a face detector, we'll first convert the image from color to grayscale. For face detection this is perfectly fine to do as there is plenty non-color specific structure in the human face for our detector to learn on.

```
In [3]: # convert to grayscale
        gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

        plt.figure(figsize=(20,10))
        plt.imshow(gray, cmap='gray')
```

```
Out[3]: <matplotlib.image.AxesImage at 0x7f483909bef0>
```

Next we load in the fully trained architecture of the face detector, found in the file `detector_architectures/ haarcascade_frontalface_default.xml`,and use it on our image to find faces!

A note on parameters

How many faces are detected is determined by the function, `detectMultiScale` which aims to detect faces of varying sizes. The inputs to this function are: `(image, scaleFactor, minNeighbors)`; you will often detect more faces with a smaller scaleFactor, and lower value for minNeighbors, but raising these values often produces better matches. Modify these values depending on your input image.

```
In [4]: # load in cascade classifier
        face_cascade = cv2.CascadeClassifier('detector_architectures/haarcascade_frontalface_def

        # run the detector on the grayscale image
        faces = face_cascade.detectMultiScale(gray, 4, 6)
```

The output of the classifier is an array of detections; coordinates that define the dimensions of a bounding box around each face. Note that this always outputs a bounding box that is square in dimension.

```
In [5]: # print out the detections found
        print ('We found ' + str(len(faces)) + ' faces in this image')
        print ("Their coordinates and lengths/widths are as follows")
        print ('=============================')
        print (faces)
```

```
We found 13 faces in this image
Their coordinates and lengths/widths are as follows
=============================
[[1295   94   96   96]
 [ 917  103   96   96]
 [1148  131   96   96]
 [ 683  149   96   96]
 [ 510  158   96   96]
 [1565  339   96   96]
 [ 588  390   96   96]
 [1157  391   96   96]
 [ 771  405   96   96]
 [ 383  414   96   96]
 [1345  411   96   96]
 [ 146  458   96   96]
 [ 996  526   96   96]]
```

Let's plot the corresponding detection boxes on our original image to see how well we've done.

```
In [ ]: img_with_detections = np.copy(image)   # make a copy of the original image to plot recta

        # loop over our detections and draw their corresponding boxes on top of our original ima
        for (x,y,w,h) in faces:
            # draw next detection as a red rectangle on top of the original image.
            # Note: the fourth element (255,0,0) determines the color of the rectangle,
            # and the final argument (here set to 5) determines the width of the drawn rectangle
            cv2.rectangle(img_with_detections,(x,y),(x+w,y+h),(255,0,0),5)

        # display the result
        plt.figure(figsize=(20,10))
        plt.imshow(img_with_detections)
```