将看到使用基于 haar 特征的级联分类器进行人脸检测的基础知识, 我们将在眼睛检测等方面进行同样的扩展。使用基于哈尔特征的级联分类器的基础对象检测是 paul viola 和 michael jones 在2001年的论文《使用可提升的简单特征级联快速对象检测》中提出的一种有效的目标检测方法。它是一种基于机器学习的方法, 在这种方法中, 级联函数是从很多正面和负面的图像中训练出来的。然后, 它用于检测其他图像中的对象。在这里, 我们将与人脸检测合作。最初, 该算法需要大量的正面图像 (人脸图像) 和负图像 (没有人脸的图像) 来训练分类器。然后我们需要从中提取特征。为此, 使用下图所示的 haar 功能。它们就像我们的卷积核。每个特征都是一个单一的值，通过从黑色矩形下的像素和减去白色矩形下的像素和得到。。



现在, 每个内核的所有可能的大小和位置都用于计算大量的要素。(试想一下, 它需要多少计算？即使是24x24 窗口, 也会产生超过16000个功能)。对于每个要素计算, 我们需要查找白色和黑色矩形下的像素之和。为了解决这个问题, 他们介绍了积分图像。无论图像有多大, 它都会将给定像素的计算简化为仅涉及四个像素的操作。很好, 不是吗？它使事情变得超快。但在我们计算的所有这些特征中, 大多数都是不相关的。例如, 请考虑下面的图像。顶行显示两个良好的功能。选择的第一个特征似乎集中在眼睛的区域往往比鼻子和脸颊的区域更黑的属性上。选择的第二个要素依赖于眼睛比鼻梁更黑的属性。但同样的窗户适用于脸颊或任何其他地方并不重要。那么, 我们如何选择..。



形象

为此，我们在所有训练图像上应用每个特征。对于每个特征，它找到将人脸分为正向和负向的最佳阈值。显然，会有错误或错误分类。我们选择具有最小错误率的特征，这意味着它们是最准确地对人脸和非人脸图像进行分类的特征。（这个过程并不像这样简单。）每个图像在开始处被赋予相等的权重。在每次分类之后，增加误分类图像的权重。然后完成相同的过程。计算新的错误率。还有新的重量。该过程继续进行，直到达到所需的精度或错误率或找到所需的特征数量为止。

最后的分类器是这些弱分类器的加权和。它之所以被称为弱分类器，是因为它不能单独对图像进行分类，而是与其他分类器一起形成一个强分类器。这篇论文说，即使200个特征也能提供95%的准确率。他们的最终设置大约有6000个特性。（想象一下从160000个特性减少到6000个特性）。这是一个巨大的收益）。

所以现在你拍一张照片。每个24x24窗口。应用6000特性。检查它是否是脸。真的。。这难道不是有点低效和费时吗？是的，是的。作者对此有很好的解决方案。

在图像中，大部分图像是非面部区域。因此，最好有一个简单的方法来检查窗口是否不是面部区域。如果不是，那么一次性丢弃它，并且不要再次处理它。取而代之的是，把注意力集中在有脸的区域。这样，我们花更多的时间检查可能的面部区域。

为此，他们引入了级联分类器的概念。不是在窗口中应用所有6000个特征，而是将这些特征分组到分类器的不同阶段并逐个应用。（通常，前几个阶段将包含非常少的特征）。如果一个窗口在第一阶段失败，丢弃它。我们不考虑它的其余特性。如果通过了，应用第二阶段的特性并继续这个过程。通过所有阶段的窗口是一个面部区域。这个计划怎么样！

作者的探测器有6000多个特征，38个阶段，前五个阶段有1、10、25、25和50个特征。（上面图像中的两个特征实际上是从Adaboost获得的最佳两个特征）。作者认为，平均每个子窗口评估6000+中的10个特征。

因此，这是一个简单的直观的解释，如何Viola-Jones脸部检测工作。阅读本文以了解更多细节，或者查看附加资源部分中的参考资料。

OpenCV中的Haar-级联检测

OpenCV附带了训练器和检测器。如果您想为汽车、飞机等任何对象训练自己的分类器，可以使用OpenCV创建一个。它的全部细节在这里给出：级联分类器训练。

这里我们将处理检测。OpenCV已经包含许多针对面部、眼睛、微笑等的预先训练的分类器。这些XML文件存储在opencv/data/haarcascades/文件夹中。让我们用OpenCV创建一个面部和眼睛检测器。

首先，我们需要加载所需的XML分类器。然后以灰度模式加载我们的输入图像（或视频）。

import numpy as np

import cv2 as cv

face\_cascade = [cv.CascadeClassifier](https://docs.opencv.org/master/d1/de5/classcv_1_1CascadeClassifier.html)('haarcascade\_frontalface\_default.xml')

eye\_cascade = [cv.CascadeClassifier](https://docs.opencv.org/master/d1/de5/classcv_1_1CascadeClassifier.html)('haarcascade\_eye.xml')

img = [cv.imread](https://docs.opencv.org/master/d4/da8/group__imgcodecs.html#ga288b8b3da0892bd651fce07b3bbd3a56)('sachin.jpg')

gray = [cv.cvtColor](https://docs.opencv.org/master/d8/d01/group__imgproc__color__conversions.html#ga397ae87e1288a81d2363b61574eb8cab)(img, cv.COLOR\_BGR2GRAY)

Now we find the faces in the image. If faces are found, it returns the positions of detected faces as Rect(x,y,w,h). Once we get these locations, we can create a ROI for the face and apply eye detection on this ROI (since eyes are always on the face !!! ).

faces = face\_cascade.detectMultiScale(gray, 1.3, 5)

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

[cv.rectangle](https://docs.opencv.org/master/d6/d6e/group__imgproc__draw.html#gac865734d137287c0afb7682ff7b3db23)(img,(x,y),(x+w,y+h),(255,0,0),2)

roi\_gray = gray[y:y+h, x:x+w]

roi\_color = img[y:y+h, x:x+w]

eyes = eye\_cascade.detectMultiScale(roi\_gray)

for (ex,ey,ew,eh) in eyes:

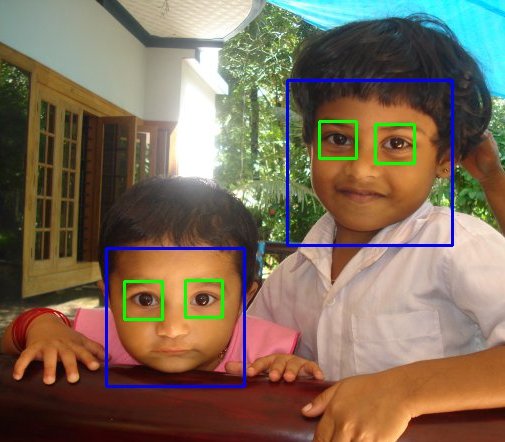
[cv.rectangle](https://docs.opencv.org/master/d6/d6e/group__imgproc__draw.html#gac865734d137287c0afb7682ff7b3db23)(roi\_color,(ex,ey),(ex+ew,ey+eh),(0,255,0),2)

[cv.imshow](https://docs.opencv.org/master/df/d24/group__highgui__opengl.html#gaae7e90aa3415c68dba22a5ff2cefc25d)('img',img)

[cv.waitKey](https://docs.opencv.org/master/d7/dfc/group__highgui.html#ga5628525ad33f52eab17feebcfba38bd7)(0)

[cv.destroyAllWindows](https://docs.opencv.org/master/d7/dfc/group__highgui.html#ga6b7fc1c1a8960438156912027b38f481)()

Result looks like below:



**image**