实验四 人脸检测与识别

1. 实验目的
2. 掌握Anaconda中应用工具库的安装及配置。
3. 熟练操作Anaconda环境中Jupyter Notebook的使用。
4. 熟悉Anaconda环境中OpenCV程序的编写及运行。
5. 实验要求
6. 熟练掌握Anaconda环境的操作。
7. 理解OpenCV 3的工作原理。
8. 将上机程序调试通过，并能独立完成课堂练习题目。
9. 实验内容

安装完成OpenCV3.3 + Python3.6 开发环境配置，并完成相关配置操作；掌握在Python编程中调用相应的库；掌握在Jupyter Notebook中的基本操作;掌握基本OpenCV python程序的编写。

1. 实验内容

**人脸检测与识别介绍：**

人脸检测属于计算机视觉的范畴，早期人们的主要研究方向是人脸识别，即根据人脸来识别人物的身份，后来在复杂背景下的人脸检测需求越来越大，人脸检测也逐渐作为一个单独的研究方向发展起来。

目前人脸检测的方法主要有两大类：基于知识和基于统计。

基于知识的方法：主要利用先验知识将人脸看作器官特征的组合，根据眼睛、眉毛、嘴巴、鼻子等器官的特征以及相互之间的几何位置关系来检测人脸。主要包括模板匹配、人脸特征、形状与边缘、纹理特性、颜色特征等方法。

基于统计的方法：将人脸看作一个整体的模式——二维像素矩阵，从统计的观点通过大量人脸图像样本构造人脸模式空间，根据相似度量来判断人脸是否存在。主要包括主成分分析与特征脸、神经网络方法、支持向量机、隐马尔可夫模型、Adaboost算法等。

本文中介绍的Haar分类器方法，包含了Adaboost算法，稍候会对这一算法做详细介绍。所谓分类器，在这里就是指对人脸和非人脸进行分类的算法，在机器学习领域，很多算法都是对事物进行分类、聚类的过程。

我们要探讨的Haar分类器实际上是Boosting算法的一个应用，Haar分类器用到了Boosting算法中的AdaBoost算法，只是把AdaBoost算法训练出的强分类器进行了级联，并且在底层的特征提取中采用了高效率的矩形特征和积分图方法，这里涉及到的几个名词接下来会具体讨论。

在2001年，Viola和Jones两位大牛发表了经典的《Rapid Object Detection using a Boosted Cascade of Simple Features》和《Robust Real-Time Face Detection》，在AdaBoost算法的基础上，使用Haar-like小波特征和积分图方法进行人脸检测，他俩不是最早使用提出小波特征的，但是他们设计了针对人脸检测更有效的特征，并对AdaBoost训练出的强分类器进行级联。这可以说是人脸检测史上里程碑式的一笔了，也因此当时提出的这个算法被称为Viola-Jones检测器。又过了一段时间，Rainer Lienhart和Jochen Maydt两位大牛将这个检测器进行了扩展，最终形成了OpenCV现在的Haar分类器。

AdaBoost是Freund和Schapire在1995年提出的算法，是对传统Boosting算法的一大提升。Boosting算法的核心思想，是将弱学习方法提升成强学习算法，也就是“三个臭皮匠顶一个诸葛亮”。

Haar分类器 = Haar-like特征 + 积分图方法 + AdaBoost +级联；

Haar分类器算法的要点如下：

使用Haar-like特征做检测。

使用积分图（Integral Image）对Haar-like特征求值进行加速。

使用AdaBoost算法训练区分人脸和非人脸的强分类器。

使用筛选式级联把强分类器级联到一起，提高准确率。

Haar分类器训练的五大步骤：

1、准备人脸、非人脸样本集；

2、计算特征值和积分图；

3、筛选出T个优秀的特征值（即最优弱分类器）；

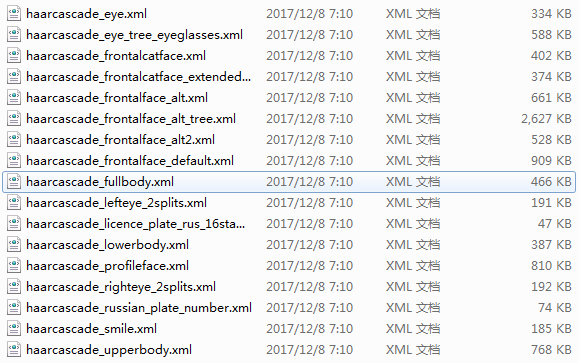
4、把这个T个最优弱分类器传给AdaBoost进行训练。

5、级联，也就是强分类器的强强联手。

在开始前，一定要记住，以20\*20窗口为例，就有78,460的特征数量，筛选出T个优秀的特征值（即最优弱分类器），然后把这个T个最优弱分类器传给AdaBoost进行训练得到一个强分类器，最后将强分类器进行级联。

**XML文件**

OpenCV 自带了训练器和检测器。如果想自己训练一个分类器来检测汽车，飞机等的话，可以使用 OpenCV 构建。其中的细节参考：Cascade Classifier Training。这里我们介绍的XML文件，就是OpenCV自带的检测器，在OpenCV 3的库文件中会包含一个文件夹haarcascades，在我的电脑上路径为D:\Anaconda\pkgs\opencv-3.3.1-py36h20b85fd\_1\Library\etc\haarcascades。在这个文件夹下包含了OpenCV的人脸检测的XML文件，这些文件可用于检测静止图像，视频和摄像头所得到图像中的人脸。除此之外还有一个文件夹是lbpcascades，它不是通过Haar特征进行人脸检测，而是采用的LBP特征。



从这些文件名可以知道这些级联适用于检测人脸、眼睛、鼻子和嘴等部位的跟踪，这些文件需要正面、直立的人体图像。

 xml文件主要保存相关的特征矩阵，以及各个弱分类器相关的信息，关于各个节点的具体含义可以参考文章[haar+adaboost结合讲解（偏重实际）](https://blog.csdn.net/playezio/article/details/80471000" \t "_blank)，这里不做过多的介绍。

### 人脸检测

在这里我们将会学习到使用级联分类器进行人脸检测。在静态图像或视频中检测人脸的操作非常相似。视频人脸检测知识从摄像头读出毎帧图像，然后采用静态图像中的人脸检测方法进行检测，当然，视频人脸检测还涉及其他概念，例如跟踪，而静态图像中人脸检测就没有这样的概念，但是它们的基本理论是一致的。

#### 1、静态图像中的人脸检测

我们首先把haarcascades文件夹复制到当前项目路径下，然后创建.py文件，代码如下：

import cv2

import numpy as np

#1.静态图像中的人脸检测

def StaticDetect(filename):

#创建一个级联分类器 加载一个 .xml 分类器文件. 它既可以是Haar特征也可以是LBP特征的分类器.

face\_cascade = cv2.CascadeClassifier('./haarcascades/haarcascade\_frontalface\_default.xml')

#加载图像

img = cv2.imread(filename)

#转换为灰度图

gray\_img = cv2.cvtColor(img,cv2.COLOR\_BGR2GRAY)

#进行人脸检测，传入scaleFactor，minNegihbors，分别表示人脸检测过程中每次迭代时图像的压缩率以及

#每个人脸矩形保留近似数目的最小值

#返回人脸矩形数组

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

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

#在原图像上绘制矩形

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

cv2.namedWindow('Face Detected！')

cv2.imshow('Face Detected！',img)

cv2.waitKey(0)

cv2.destroyAllWindows()

#2、视频中的人脸检测

def DynamicDetect():

'''

打开摄像头，读取帧，检测帧中的人脸，扫描检测到的人脸中的眼睛，对人脸绘制蓝色的矩形框，对人眼绘制绿色的矩形框

'''

#创建一个级联分类器 加载一个 .xml 分类器文件. 它既可以是Haar特征也可以是LBP特征的分类器.

face\_cascade = cv2.CascadeClassifier('./haarcascades/haarcascade\_frontalface\_default.xml')

eye\_cascade = cv2.CascadeClassifier('./haarcascades/haarcascade\_eye.xml')

#打开摄像头

camera = cv2.VideoCapture(0)

cv2.namedWindow('Dynamic')

while(True):

#读取一帧图像

ret,frame = camera.read()

#判断图片读取成功？

if ret:

gray\_img = cv2.cvtColor(frame,cv2.COLOR\_BGR2GRAY)

#人脸检测

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

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

#在原图像上绘制矩形

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

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

#眼睛检测

eyes = eye\_cascade.detectMultiScale(roi\_gray,1.03,5,0,(40,40))

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

cv2.rectangle(frame,(ex+x,ey+y),(x+ex+ew,y+ey+eh),(0,255,0),2)

cv2.imshow('Dynamic',frame)

#如果按下q键则退出

if cv2.waitKey(100) & 0xff == ord('q') :

break

camera.release()

cv2.destroyAllWindows()

if \_\_name\_\_=='\_\_main\_\_':

#filename = 'timg.jpg'

#StaticDetect(filename)

DynamicDetect()

我们来分析一下StaticDetect函数，首先创建一个级联分类器对象，然后加载xml检测器，用来进行人脸检测。

#创建一个级联分类器 加载一个 .xml 分类器文件. 它既可以是Haar特征也可以是LBP特征的分类器.

face\_cascade = cv2.CascadeClassifier('./haarcascades/haarcascade\_frontalface\_default.xml')

然后加载图片文件，并将其转换为灰度图像，因为人脸检测需要这样的色彩空间。

#加载图像

img = cv2.imread(filename)

#转换为灰度图

gray\_img = cv2.cvtColor(img,cv2.COLOR\_BGR2GRAY)

接下来进行人脸检测，需要传入scaleFactor和minNeighbors参数，它们分别表示人脸检测过程中每次迭代时图像的压缩率以及每个人脸矩形保留近似数目的最小值。然后函数返回人脸矩阵数组。我们利用cv2.rectangle函数在原图中把矩形绘制出来。

#进行人脸检测，传入scaleFactor，minNegihbors，分别表示人脸检测过程中每次迭代时图像的压缩率以及

#每个人脸矩形保留近似数目的最小值

#返回人脸矩形数组

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

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

#在原图像上绘制矩形

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

cv2.namedWindow('Face Detected！')

cv2.imshow('Face Detected！',img)

cv2.waitKey(0)

cv2.destroyAllWindows()

运行结果如下，我们可以看到所有的人脸都检测出来了，但是其中还有一个误检测：

1. 实验总结

完成人脸识别程序的调试，并完成实验报告总结的编写。