# 任务4 摄像头图像捕捉和人脸检测

## 任务描述

**内 容**：从USB摄像头捕捉图像；人脸检测。

**学 时**：2

**知识点**：使用opencv库通过摄像头拍摄照片并保存、使用dlib库检测人脸

**重点**：通过摄像头拍摄照片并保存、使用dlib库检测人脸

**难点**：通过摄像头拍摄照片并保存、使用dlib库检测人脸

## 授课思路

从本任务开始同学们开始编写计算机视觉的程序。

本次任务在任务实现中提供完整代码。学生通过**代码**和**网上查找资料**进行学习。请同学们按照任务实现学会这2个知识点。网上查找资料学习的是dlib库人脸检测的技术原理，然后通过代码进行实践。

## 任务指导

* 1. 硬件准备：USB摄像头；
  2. 使用opencv库通过摄像头拍摄照片并保存；

业务场景：通过摄像头拍摄100张图片，并保存

* 1. 理解haarcascade人脸检测的原理；
  2. 理解dlib库的hog人脸检测的原理（课堂上不做要求，课余时间研究）；
  3. 使用dlib库检测人脸。

业务场景：通过摄像头实时捕捉画面，如果画面中检测到人脸，就用矩形框出来。

## 任务实现

1. 硬件准备：USB摄像头；

同学们如果用的是笔记本，只需要确保笔记本摄像头能使用即可；同学们如果使用的是台式机，请准别一个USB摄像头。

1. 使用opencv库通过摄像头拍摄照片并保存

本次子任务的完整代码放在 **任务源代码/任务4. 摄像头图像捕捉和人脸检测/1.使用opencv库通过摄像头拍摄照片并保存** 中。

本次子任务的目录结构为:



**images**是保存图片的目录，**captureandsavephotos.py**是主程序。

打开**captureandsavephotos.py**，添加如下代码：

# -\*- coding: utf-8 -\*-

'''

使用Web摄像头 (USB摄像头)捕捉图像并保存

'''

import cv2

import time

cap = cv2.VideoCapture(0)

cap.set(0,640) # set Width (the first parameter is property\_id)

cap.set(1,480) # set Height

time.sleep(2)

for i in range(100):# 拍100张图片就结束

ret, img = cap.read()

cv2.imshow('img', img)

cv2.imwrite('images/%d.jpg' %(i), img)

# Press 'ESC' for exiting video

k = cv2.waitKey(100) & 0xff

if k == 27:

break

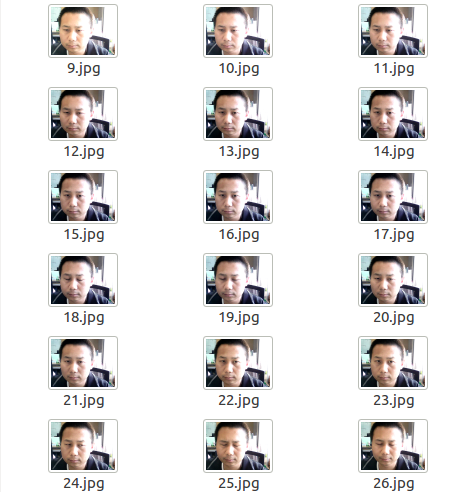
cap.release()

cv2.destroyAllWindows()

执行python文件，opencv弹出框显示实时画面。



程序结束后，images文件夹生成了100张图像。



1. 理解haarcascade人脸检测的原理

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。

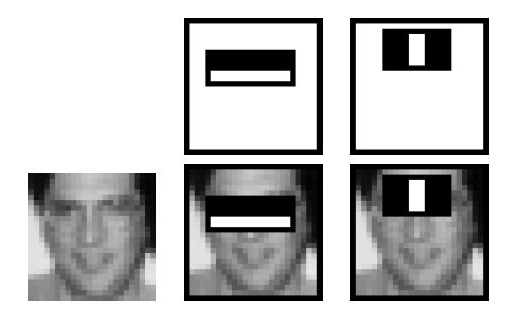
**1）Haar-like特征**

Haar(哈尔)特征分为三类：边缘特征、线性特征、中心特征和对角线特征，组合成特征模板。特征模板内有白色和黑色两种矩形，并定义该模板的特征值为白色矩形像素和减去黑色矩形像素和。Haar特征值反映了图像的灰度变化情况。例如：脸部的一些特征能由矩形特征简单的描述，如：眼睛要比脸颊颜色要深，鼻梁两侧比鼻梁颜色要深，嘴巴比周围颜色要深等。但矩形特征只对一些简单的图形结构，如边缘、线段较敏感，所以只能描述特定走向（水平、垂直、对角）的结构。



对于图中的A, B和D这类特征，特征数值计算公式为：v=Σ白-Σ黑，而对于C来说，计算公式如下：v=Σ白-2\*Σ黑；之所以将黑色区域像素和乘以2，是为了使两种矩形区域中像素数目一致。我们希望当把矩形放到人脸区域计算出来的特征值和放到非人脸区域计算出来的特征值差别越大越好，这样就可以用来区分人脸和非人脸。

通过改变特征模板的大小和位置，可在图像子窗口中穷举出大量的特征。上图的特征模板称为“特征原型”；特征原型在图像子窗口中扩展（平移伸缩）得到的特征称为“矩形特征”；矩形特征的值称为“特征值”。



上图中两个矩形特征，表示出人脸的某些特征。比如中间一幅表示眼睛区域的颜色比脸颊区域的颜色深，右边一幅表示鼻梁两侧比鼻梁的颜色要深。同样，其他目标，如眼睛等，也可以用一些矩形特征来表示。使用特征比单纯地使用像素点具有很大的优越性，并且速度更快。

矩形特征可位于图像任意位置，大小也可以任意改变，所以矩形特征值是矩形模版类别、矩形位置和矩形大小这三个因素的函数。故类别、大小和位置的变化，使得很小的检测窗口含有非常多的矩形特征，如：在24\*24像素大小的检测窗口内矩形特征数量可以达到16万个。这样就有两个问题需要解决了：（1）如何快速计算那么多的特征？---积分图大显神通；（2）哪些矩形特征才是对分类器分类最有效的？---如通过AdaBoost算法来训练。

**2）Haar-like特征的计算—积分图**

积分图就是只遍历一次图像就可以求出图像中所有区域像素和的快速算法，大大的提高了图像特征值计算的效率。

积分图主要的思想是将图像从起点开始到各个点所形成的矩形区域像素之和作为一个数组的元素保存在内存中，当要计算某个区域的像素和时可以直接索引数组的元素，不用重新计算这个区域的像素和，从而加快了计算（这有个相应的称呼，叫做动态规划算法）。积分图能够在多种尺度下，使用相同的时间（常数时间）来计算不同的特征，因此大大提高了检测速度。

积分图是一种能够描述全局信息的矩阵表示方法。积分图的构造方式是位置（i,j）（i,j）处的值ii(i,j)ii(i,j)是原图像(i,j)(i,j)左上角方向所有像素f(k,l)f(k,l)的和：

ii(i,j)=∑k≤i,l≤jf(k,l)ii(i,j)=∑k≤i,l≤jf(k,l)

积分图构建算法：

1、用s(i,j)s(i,j)表示行方向的累加和，初始化s(i,−1)=0s(i,−1)=0；

2、使用ii(i,j)ii(i,j)表示一个积分图像，初始化ii(−1,i)ii(−1,i)=0；

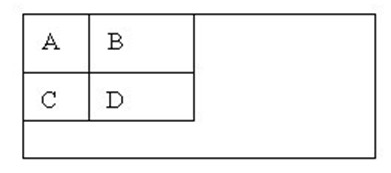
3、逐行扫描图像，递归计算每个像素(i,j)(i,j)行方向的累加和s(i,j)s(i,j)和积分图像ii(i,j)ii(i,j)的值：

s(i,j)=s(i,j−1)+f(i,j)s(i,j)=s(i,j−1)+f(i,j)

ii(i,j)=ii(i−1,j)+s(i,j)ii(i,j)=ii(i−1,j)+s(i,j)

4、扫描图像一遍，当到达图像右下角像素时，积分图像iiii就构建好了。

积分图构造好之后，图像中任何矩阵区域像素累加和都可以通过简单运算得到如图所示：



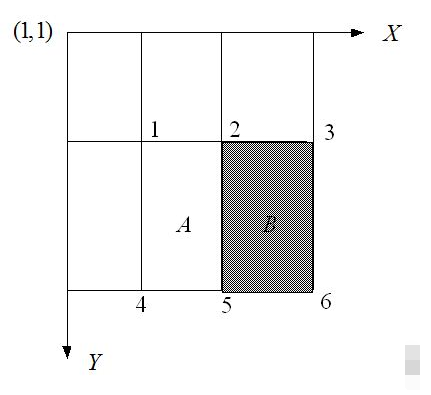
设D的四个顶点分别为α,β,γ,δ则D的像素和可以表示位

Dsum=ii(α)+ii(β)−(ii(γ)+ii(δ))Dsum=ii(α)+ii(β)−(ii(γ)+ii(δ))

而Haar-like特征值无非就是两个矩阵像素和的差，同样可以在常数时间内完成。

**3）计算Haar特征值**

上面已经知道，一个区域的像素值的和，可以由该区域的端点的积分图来计算。由前面特征模板的特征值的定义可以推出，矩形特征的特征值可以由特征端点的积分图计算出来。以A矩形特征为例，如下图，使用积分图计算其特征值：



该矩形特征的特征值，由定义，为区域A的像素值减去区域B的像素值。

区域A的像素值：

ii(5)+ii(1)−ii(2)−ii(4)ii(5)+ii(1)−ii(2)−ii(4)

区域B的像素值：

ii(6)+ii(2)−ii(5)−ii(3)ii(6)+ii(2)−ii(5)−ii(3)

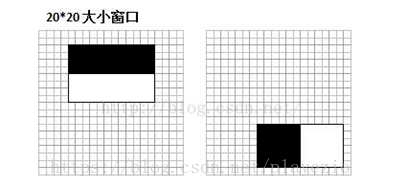
所以：该矩形特征的特征值

ii(5)+ii(1)−ii(2)−ii(4)−[ii(6)+ii(2)−ii(5)−ii(3)]ii(5)+ii(1)−ii(2)−ii(4)−[ii(6)+ii(2)−ii(5)−ii(3)]

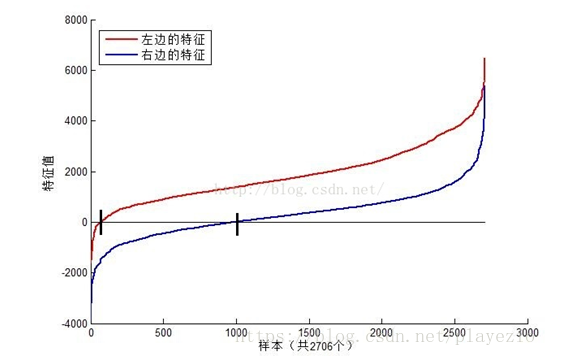
=[ii(5)−ii(4)]+[ii(3)−ii(2)]−[ii(2)−ii(1)]−[ii(6)−ii(5)]=[ii(5)−ii(4)]+[ii(3)−ii(2)]−[ii(2)−ii(1)]−[ii(6)−ii(5)]

所以，矩形特征的特征值，只与特征矩形的端点的积分图有关，而与图像的坐标无关。通过计算特征矩形的端点的积分图，再进行简单的加减运算，就可以得到特征值，正因为如此，特征的计算速度大大提高，也提高了目标的检测速度。

了解了特征值的计算之后，我们来看看不同的特征值的含义是什么。我们选取MIT人脸库中2706个大小为20\*20的人脸正样本图像，计算如下图所示的Haar特征：



左边对应的人眼区域，右边无具体意义。



可以看到，图中2个不同Haar特征在同一组样本中具有不同的特征值分布，左边特征计算出的特征值基本都大于0（对样本的区分度大），而右边特征的特征值基本均匀分布于0两侧（对样本的区分度小）。所以，正是由于样本中Haar特征值分布不均匀，导致了不同Haar特征分类效果不同。显而易见，对正负样本区分度越大的特征分类效果越好，即红色曲线对应图中的的左边Haar特征分类效果好于右边Haar特征。

那么看到这里，应该理解了下面2个问题：

（1）在检测窗口通过平移+缩放可以产生一系列Haar特征，这些特征由于位置和大小不同，分类效果也不同；

（2）通过计算Haar特征的特征值，可以有将图像矩阵映射为1维特征值，有效实现了降维。

**4）Haar特征值归一化**

从上图我们可以发现，仅仅一个12\*8大小的Haar特征计算出的特征值变化范围从-2000~+6000，跨度非常大。这种跨度大的特性不利于量化评定特征值，所以需要进行“归一化”，压缩特征值范围。假设当前检测窗口中的图像像素为i(x,y)i(x,y)，当前检测窗口为w∗hw∗h大小（例如上图中为20\*20大小），OpenCV采用如下方式“归一化”：

1、计算检测窗口中图像的灰度值和灰度值平方和：

sum=∑i(x,y)sum=∑i(x,y)

sqsum=∑i2(x,y)sqsum=∑i2(x,y)

 2、计算平均值：

mean=sumw∗hmean=sumw∗h

sqmean=sqsumw∗hsqmean=sqsumw∗h

3、计算归一化因子：

varNormFactor=sqmean−mean2−−−−−−−−−−−−−√varNormFactor=sqmean−mean2

4、归一化特征值：

normValue=featureValuevarNormFactornormValue=featureValuevarNormFactor

之后使用归一化的特征值normValuenormValue与阈值对比。

**5）Adaboost级联分类器**

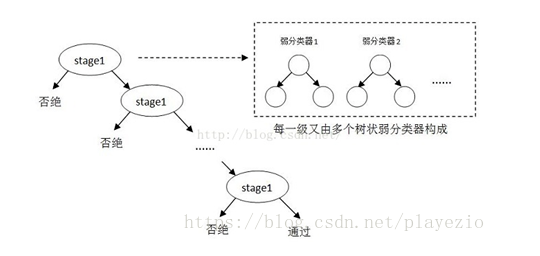
前面几块内容我们分析了Haar特征，积分图、特征值计算。这里则主要分析一下2个内容：

（1）OpenCV中的Adaboost级联分类器的结构，包括强分类器和弱分类器的形式；

（2）OpenCV自带的XML分类器中各项参数，如internalNodes和leafValues标签里面的一大堆数字的意义。

1、级联分类器

在[集成学习值Adaboost算法原理和代码小结(转载)](https://www.cnblogs.com/zyly/p/9416263.html)小节中我们已经介绍过了Adboost分类器，这里我们会介绍一下Adaboost级联分类器。级联分类模型是树状结构可以用下图表示：



其中每一个stage都代表一级强分类器。当检测窗口通过所有的强分类器时才被认为是正样本，否则拒绝。实际上，不仅强分类器是树状结构，强分类器中的每一个弱分类器也是树状结构。由于每一个强分类器对负样本的判别准确度非常高，所以一旦发现检测到的目标位负样本，就不在继续调用下面的强分类器，减少了很多的检测时间。因为一幅图像中待检测的区域很多都是负样本，这样由级联分类器在分类器的初期就抛弃了很多负样本的复杂检测，所以级联分类器的速度是非常快的；只有正样本才会送到下一个强分类器进行再次检验，这样就保证了最后输出的正样本的伪正(false positive)的可能性非常低。

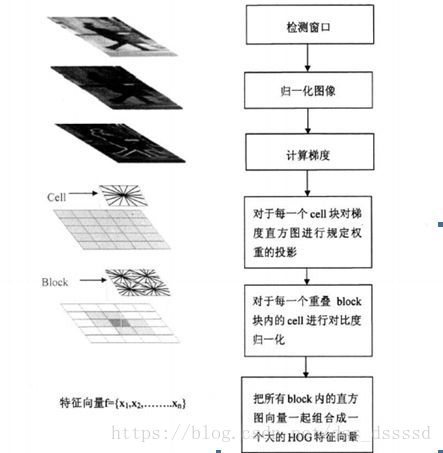
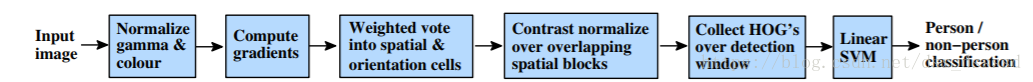
参考资料：

<https://www.cnblogs.com/zyly/p/9410563.html>

<https://www.quora.com/How-integral-image-is-used-in-image-processing-and-how-improves-the-computation-time>

<https://blog.csdn.net/wutao1530663/article/details/78294349>

1. 理解dlib库的hog人脸检测的原理



参考资料：

<https://blog.csdn.net/dss_dssssd/article/details/82663779>

<https://www.cnblogs.com/zhehan54/p/6723956.html>

1. 使用dlib库检测人脸

本次子任务的完整代码放在 **任务源代码/任务4. 摄像头图像捕捉和人脸检测/2.使用dlib库检测人脸** 中。

本次子任务只有1个python文件，不需要目录或者package。

新建一个module，命名为**facedetectionwithdlib.py**。添加如下代码：

# -\*- coding: utf-8 -\*-

'''

使用dlib实现人脸检测

'''

import face\_recognition

import cv2

import time

# 超参数

detection\_method = 'hog' # either 'hog' or 'cnn'. default is hog.

# 初始化摄像头

cap = cv2.VideoCapture(0)

cap.set(0,640) # set Width (the first parameter is property\_id)

cap.set(1,480) # set Height

time.sleep(2)

while True:# 拍100张图片就结束

ret, img = cap.read()

# 人脸检测不依赖色彩，所以先把人脸图像转成灰度图像

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

face\_locations = face\_recognition.face\_locations(

gray, number\_of\_times\_to\_upsample=1,

model = detection\_method)

# 人脸位置

for (top, right, bottom, left) in face\_locations:

cv2.rectangle(img, (left, top), (right, bottom),

(0, 0, 255), 2)

cv2.rectangle(gray, (left, top), (right, bottom),

(0, 0, 255), 2)

cv2.imshow('origin image', img)

cv2.imshow('grayscale image', gray)

# Press 'ESC' for exiting video

k = cv2.waitKey(100) & 0xff

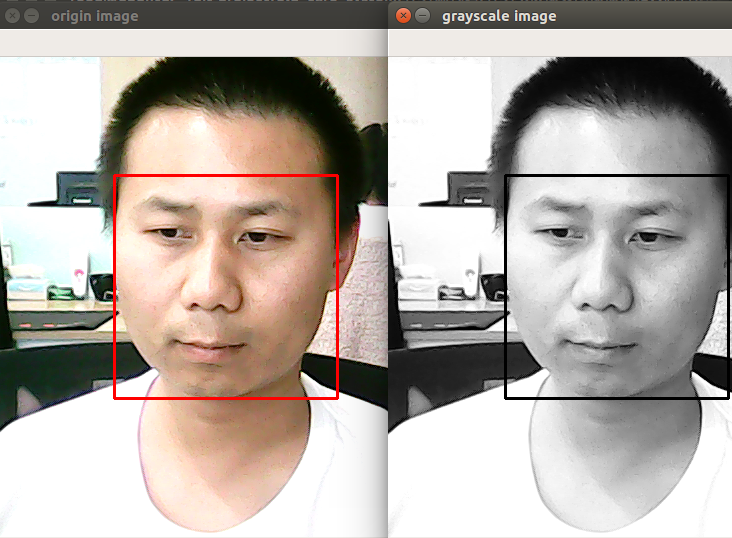
if k == 27:

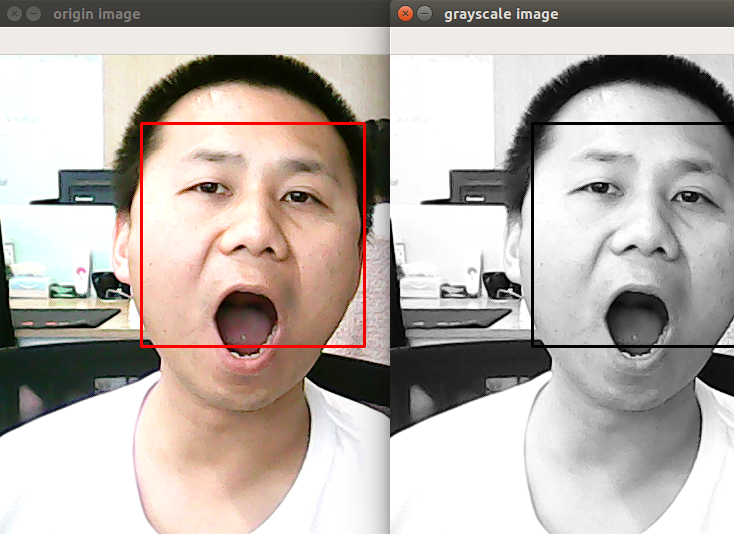
break

cap.release()

cv2.destroyAllWindows()

运行程序，弹出opencv画面，实时捕捉画面。一旦检测到人脸就会使用矩形框出来。





请同学们做一些表情，如抬头、低头、张嘴、眨眼、微笑等动作，看看人脸检测率如何。

# 任务5 老人/员工/义工人脸图像采集

## 任务描述

**内 容**：为老人、员工、义工录入信息时，采集人脸图像。

**学 时**：2

**知识点**：实现符合业务场景的人脸图像采集

**重点**：实现符合业务场景的人脸图像采集

**难点**：实现符合业务场景的人脸图像采集

## 授课思路

上个任务中，同学们学会了从USB摄像头捕捉图像，也学会了使用dlib库在图像中把脸框起来。

本次任务的目的，就是让同学们灵活应用上个任务所学的知识，实现符合业务场景的人脸图像采集。

## 任务指导

业务场景如下：

1)当人没有把脸置于框内时，系统语音提示：“没有检测到人脸”。

当系统发现框内有大于2张脸的时候，系统语音提示：“发现多张人脸”。

当一切正常时，系统语音提示：“可以开始采集图像了”。

2）系统随即又语音提示：“1))请眨眼”，继续不断保存15张图片（系统语音提示的同时，请眨眼这3个字也要在页面显示出来）。接着，系统语音提示：“2))请张嘴”，继续不断保存15张图片（系统语音提示的同时，请张嘴这3个字也要在页面显示出来）。以此类推执行下列剩余的5个动作：3))请笑一笑，4))请抬头，5))请低头，6))请看左边，7))请看右边。

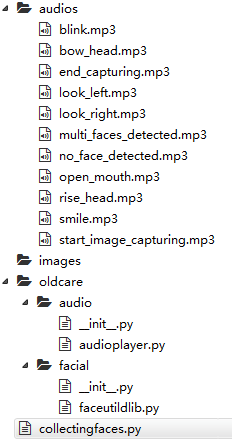
结束时系统语音提示：“采集完毕”。

3）调用程序时需要传入ID号，所有这个人的图像都存放在以这个ID号为命名的目录下。

## 任务实现

本次子任务的完整代码放在 **任务源代码/任务5.老人员工义工人脸图像采集** 中。

本任务的目录层级结构如下图所示：



在根目录下创建一个目录，命名为**audios**。里面存放着所有用到的音频文件。音频文件同学们想一想如何生成（提示：可以自己录制，也可以使用文字转语音的API）。

在根目录下创建一个目录，命名为**images**。人脸图像将存储在这个目录下面。

在根目录下新建一个package，命名为**oldcare**。

在oldcare中新建一个package，命名为**audio**。在audio这个package里面新建一个module，命名为audioplayer.py。该模块的功能为播放音频文件。代码如下：

# -\*- coding: utf-8 -\*-

'''

audio player

'''

# import library

from subprocess import call

# play audio

def play\_audio(audio\_name):

try:

call('mpg321 ' + audio\_name, shell=True) # use mpg321 player

except KeyboardInterrupt as e:

print(e)

finally:

pass

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

pass

播放音频的原理比较简单，对于linux系统而言就是调用mpg321这个软件。

在oldcare这个package里面再新建一个package，命名为**facial**。在facial这个package里面新建一个module，命名为faceutildlib.py。这个模块的功能为**人脸检测**、**人脸识别**、**训练人脸识别模型**。当前任务只关注**人脸检测**的功能。代码如下：

# -\*- coding: utf-8 -\*-

'''

使用dlib实现人脸检测

'''

import face\_recognition

import cv2

class FaceUtil:

# 超参数

detection\_method = 'hog' # either 'hog' or 'cnn'. default is hog.

# face detection

def get\_face\_location(self, image):

face\_location\_list = []

gray = cv2.cvtColor(image, cv2.COLOR\_BGR2GRAY)

face\_locations = face\_recognition.face\_locations(

gray, number\_of\_times\_to\_upsample=1,

model = self.detection\_method)

# 人脸位置

for (top, right, bottom, left) in face\_locations:

face\_location\_list.append((left, top, right, bottom))

return face\_location\_list

打开facial这个package里面的\_\_init\_\_.py，添加1行代码：

from .faceutildlib import FaceUtil

在根目录下新建一个module，命名为**collectingfaces.py**。此为人脸图像采集的主程序。代码如下：

# -\*- coding: utf-8 -\*-

'''

图像采集程序-人脸检测

由于外部程序需要调用它，所以不能使用相对路径

用法：

python collectingfaces.py --id 106 --imagedir /home/reed/git-project/

old\_care\_system/任务源代码/任务5.老人员工义工人脸图像采集/images

'''

import argparse

from oldcare.facial import FaceUtil

from oldcare.audio import audioplayer

from PIL import Image, ImageDraw, ImageFont

import cv2

import numpy as np

import os

import shutil

import time

# 全局参数

audio\_dir = '/home/reed/git-project/old\_care\_system/任务源代码/

任务5.老人员工义工人脸图像采集/audios'

# 控制参数

error = 0

start\_time = None

limit\_time = 2 # 2 秒

# 传入参数

ap = argparse.ArgumentParser()

ap.add\_argument("-ic", "--id", required=True,

help="")

ap.add\_argument("-id", "--imagedir", required=True,

help="")

args = vars(ap.parse\_args())

action\_list = ['blink', 'open\_mouth','smile','rise\_head','bow\_head',

'look\_left','look\_right']

action\_map = {'blink':'请眨眼', 'open\_mouth':'请张嘴',

'smile':'请笑一笑', 'rise\_head':'请抬头',

'bow\_head':'请低头', 'look\_left':'请看左边',

'look\_right':'请看右边'}

# 设置摄像头

cam = cv2.VideoCapture(0)

cam.set(3, 640) # set video widht

cam.set(4, 480) # set video height

faceutil = FaceUtil()

counter = 0

while True:

counter += 1

\_, image =cam.read()

if counter <=10: # 放弃前10帧

continue

image = cv2.flip(image, 1)

if error == 1:

end\_time = time.time()

difference = end\_time - start\_time

print(difference)

if difference >= limit\_time:

error = 0

face\_location\_list = faceutil.get\_face\_location(image)

for (left, top, right, bottom) in face\_location\_list:

cv2.rectangle(image, (left, top), (right, bottom),

(0, 0, 255), 2)

cv2.imshow('Collecting Faces', image) # show the image

# Press 'ESC' for exiting video

k = cv2.waitKey(100) & 0xff

if k == 27:

break

face\_count = len(face\_location\_list)

if error == 0 and face\_count == 0: # 没有检测到人脸

print('[WARNING] 没有检测到人脸')

audioplayer.play\_audio(os.path.join(audio\_dir,

'no\_face\_detected.mp3'))

error = 1

start\_time = time.time()

elif error == 0 and face\_count == 1: # 可以开始采集图像了

print('[INFO] 可以开始采集图像了')

audioplayer.play\_audio(os.path.join(audio\_dir,

'start\_image\_capturing.mp3'))

break

elif error == 0 and face\_count > 1: # 检测到多张人脸

print('[WARNING] 检测到多张人脸')

audioplayer.play\_audio(os.path.join(audio\_dir,

'multi\_faces\_detected.mp3'))

error = 1

start\_time = time.time()

else:

pass

# 新建目录

if os.path.exists(os.path.join(args['imagedir'],args['id'])):

shutil.rmtree(os.path.join(args['imagedir'],args['id']),True)

os.mkdir(os.path.join(args['imagedir'],args['id']))

# 开始采集人脸

for action in action\_list:

audioplayer.play\_audio(os.path.join(audio\_dir,action+'.mp3'))

action\_name = action\_map[action]

counter = 1

for i in range(15):

print('%s-%d' %(action\_name, i))

\_, img\_OpenCV =cam.read()

img\_OpenCV = cv2.flip(img\_OpenCV, 1)

origin\_img = img\_OpenCV.copy() # 保存时使用

face\_location\_list = faceutil.get\_face\_location(img\_OpenCV)

for (left, top, right, bottom) in face\_location\_list:

cv2.rectangle(img\_OpenCV, (left, top),

(right, bottom), (0, 0, 255), 2)

img\_PIL = Image.fromarray(cv2.cvtColor(img\_OpenCV,

cv2.COLOR\_BGR2RGB))

draw = ImageDraw.Draw(img\_PIL)

draw.text((int(image.shape[1]/2), 30), action\_name,

font=ImageFont.truetype('NotoSansCJK-Black.ttc',40),

fill=(255,0,0)) # linux

# 转换回OpenCV格式

img\_OpenCV = cv2.cvtColor(np.asarray(img\_PIL),

cv2.COLOR\_RGB2BGR)

cv2.imshow('Collecting Faces', img\_OpenCV) # show the image

image\_name = os.path.join(args['imagedir'],args['id'],

action+'\_'+str(counter)+'.jpg')

cv2.imwrite(image\_name, origin\_img)

# Press 'ESC' for exiting video

k = cv2.waitKey(100) & 0xff

if k == 27:

break

counter += 1

# 结束

print('[INFO] 采集完毕')

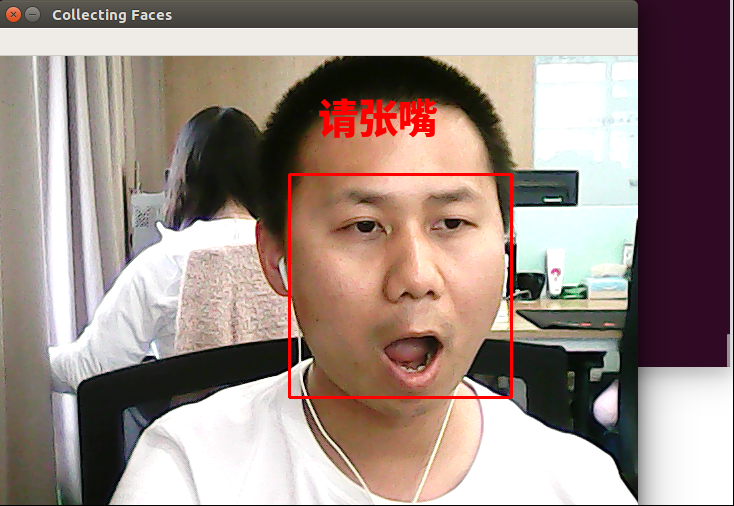
audioplayer.play\_audio(os.path.join(audio\_dir,'end\_capturing.mp3'))

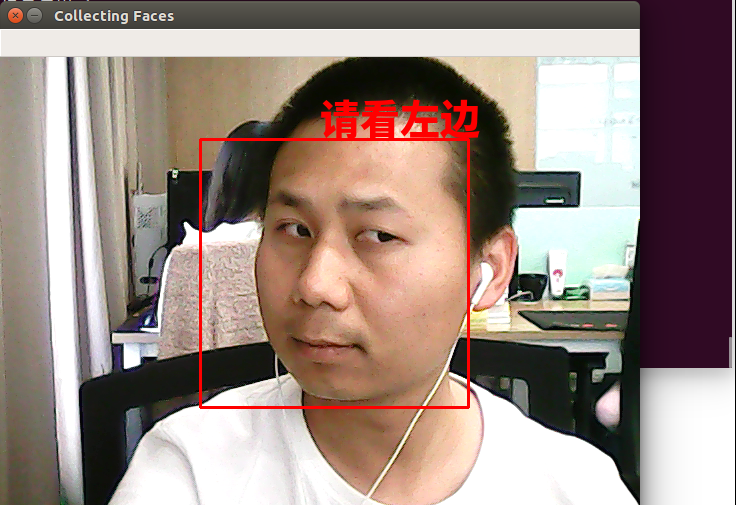
# 释放全部资源

cam.release()

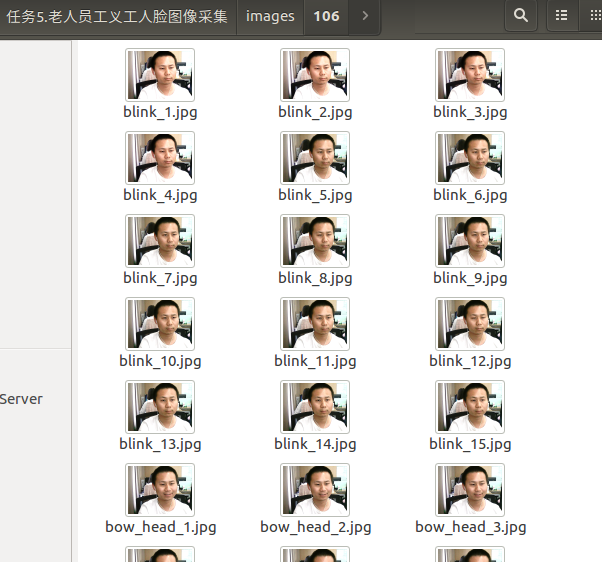
cv2.destroyAllWindows()

执行**python collectingfaces.py --id 106 --imagedir /home/reed/git-project/old\_care\_system/任务源代码/任务5.老人员工义工人脸图像采集/images**运行程序收集老人/员工/义工的人脸图像。--id指用户ID,--imagedir指保存图像的目录。程序运行效果如下图：





程序运行结束后，images目录下会多一个以106命名的目录。106目录里存放着图像文件，如下图所示：



在系统的应用场景中，Web前端会调用该程序，然后该程序完成人脸的采集。Id和imagedir的值便是由Web端传入的。这也解释了这里为什么要用绝对路径，因为外部程序需要调用它。

# 任务6 理解人脸识别原理

## 任务描述

**内 容**：理解hog算法，理解人脸检测的原理、理解人脸识别的原理

**学 时**：2

**知识点**：理解hog算法、理解人脸识别的原理

**重点**：理解hog算法、理解人脸识别的原理

**难点**：理解hog算法、理解人脸识别的原理

## 授课思路

本任务需要学生往常查找资料进行学习。之后师生互动交流。

## 任务指导

1. 完成对hog方式进行人脸检测的理解
2. 查找资料理解hog方式进行人脸识别

## 任务实现

1. 完成对hog方式进行人脸检测的理解

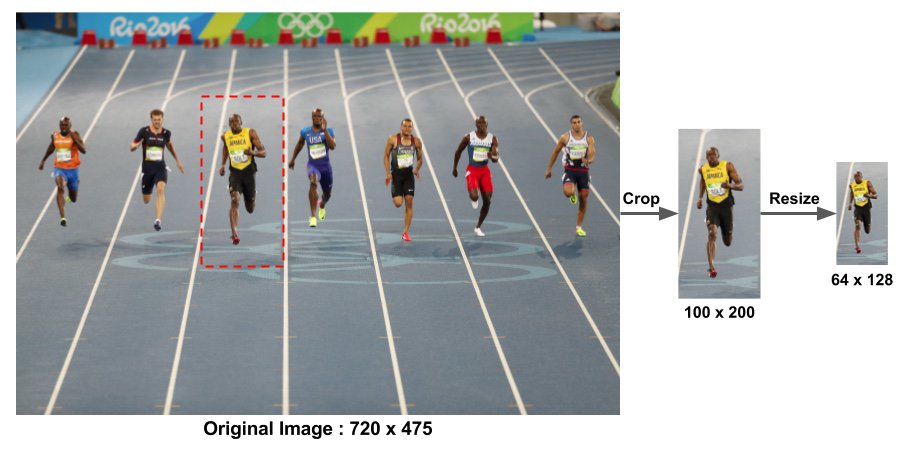
How to calculate Histogram of Oriented Gradients ?

In this section, we will go into the details of calculating the HOG feature descriptor. To illustrate each step, we will use a patch of an image.

**Step 1 : Preprocessing**

As mentioned earlier HOG feature descriptor used for pedestrian detection is calculated on a 64×128 patch of an image. Of course, an image may be of any size. Typically patches at multiple scales are analyzed at many image locations. The only constraint is that the patches being analyzed have a fixed aspect ratio. In our case, the patches need to have an aspect ratio of 1:2. For example, they can be 100×200, 128×256, or 1000×2000 but not 101×205.

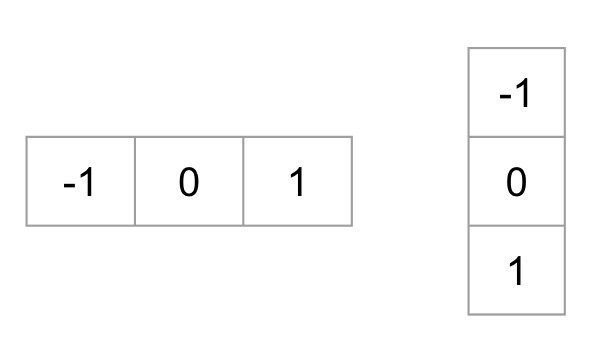
To illustrate this point I have shown a large image of size 720×475. We have selected a patch of size 100×200 for calculating our HOG feature descriptor. This patch is cropped out of an image and resized to 64×128. Now we are ready to calculate the HOG descriptor for this image patch.

[](https://www.learnopencv.com/wp-content/uploads/2016/11/hog-preprocessing.jpg)

The paper by Dalal and Triggs also mentions gamma correction as a preprocessing step, but the performance gains are minor and so we are skipping the step.

**Step 2 : Calculate the Gradient Images**

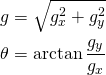
To calculate a HOG descriptor, we need to first calculate the horizontal and vertical gradients; after all, we want to calculate the histogram of gradients. This is easily achieved by filtering the image with the following kernels.

[](https://www.learnopencv.com/wp-content/uploads/2016/11/gradient-kernels.jpg)

We can also achieve the same results, by using **Sobel** operator in OpenCV with kernel size 1.

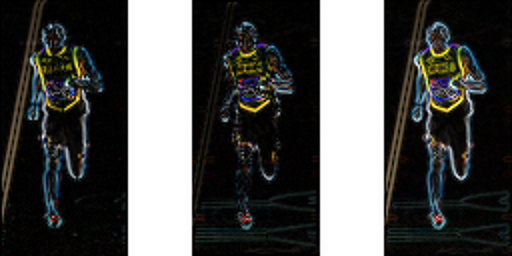
|  |
| --- |
| # Python gradient calculation  # Read image  im = cv2.imread('bolt.png')  im = np.float32(im) / 255.0    # Calculate gradient  gx = cv2.Sobel(img, cv2.CV\_32F, 1, 0, ksize=1)  gy = cv2.Sobel(img, cv2.CV\_32F, 0, 1, ksize=1) |

Next, we can find the magnitude and direction of gradient using the following formula



|  |
| --- |
| # Python Calculate gradient magnitude and direction ( in degrees )  mag, angle = cv2.cartToPolar(gx, gy, angleInDegrees=True) |

The figure below shows the gradients.

[](https://www.learnopencv.com/wp-content/uploads/2016/11/gradients.png)Left : Absolute value of x-gradient. Center : Absolute value of y-gradient. Right : Magnitude of gradient.

Notice, the x-gradient fires on vertical lines and the y-gradient fires on horizontal lines. The magnitude of gradient fires where ever there is a sharp change in intensity. None of them fire when the region is smooth. I have deliberately left out the image showing the direction of gradient because direction shown as an image does not convey much.

The gradient image removed a lot of non-essential information ( e.g. constant colored background ), but highlighted outlines. In other words, you can look at the gradient image and still easily say there is a person in the picture.

At every pixel, the gradient has a magnitude and a direction. For color images, the gradients of the three channels are evaluated ( as shown in the figure above ). The magnitude of gradient at a pixel is the maximum of the magnitude of gradients of the three channels, and the angle is the angle corresponding to the maximum gradient.

**Step 3 : Calculate Histogram of Gradients in 8×8 cells**

[](https://www.learnopencv.com/wp-content/uploads/2016/11/hog-cells.png)8×8 cells of HOG. Image is scaled by 4x for display.

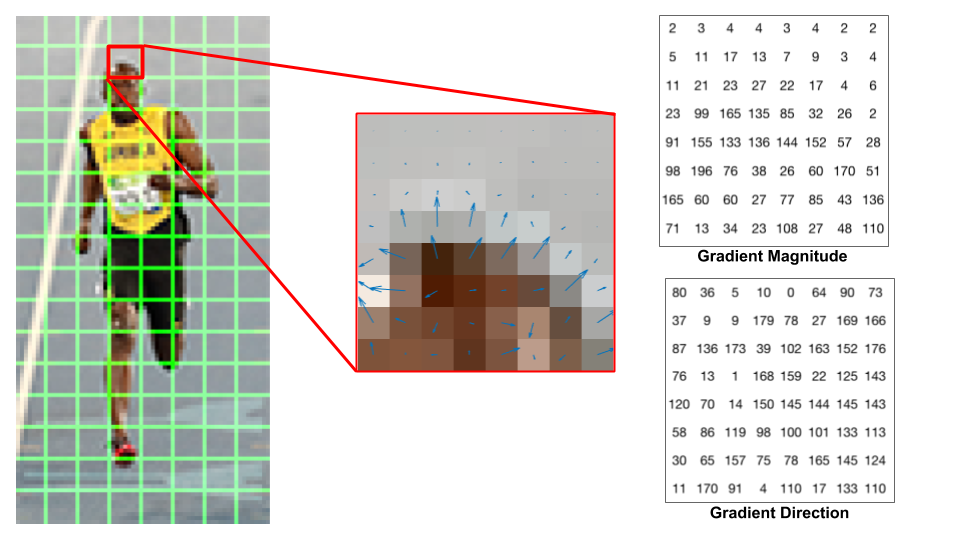
In this step, the image is divided into 8×8 cells and a histogram of gradients is calculated for each 8×8 cells.

We will learn about the histograms in a moment, but before we go there let us first understand why we have divided the image into 8×8 cells. One of the important reasons to use a feature descriptor to describe a patch of an image is that it provides a compact representation. An 8×8 image patch contains 8x8x3 = 192 pixel values. The gradient of this patch contains 2 values ( magnitude and direction ) per pixel which adds up to 8x8x2 = 128 numbers. By the end of this section we will see how these 128 numbers are represented using a 9-bin histogram which can be stored as an array of 9 numbers. Not only is the representation more compact, calculating a histogram over a patch makes this represenation more robust to noise. Individual graidents may have noise, but a histogram over 8×8 patch makes the representation much less sensitive to noise.

But why 8×8 patch ? Why not 32×32 ? It is a design choice informed by the scale of features we are looking for. HOG was used for pedestrian detection initially. 8×8 cells in a photo of a pedestrian scaled to 64×128 are big enough to capture interesting features ( e.g. the face, the top of the head etc. ).

The histogram is essentially a vector ( or an array ) of 9 bins ( numbers ) corresponding to angles 0, 20, 40, 60 … 160.

Let us look at one 8×8 patch in the image and see how the gradients look.

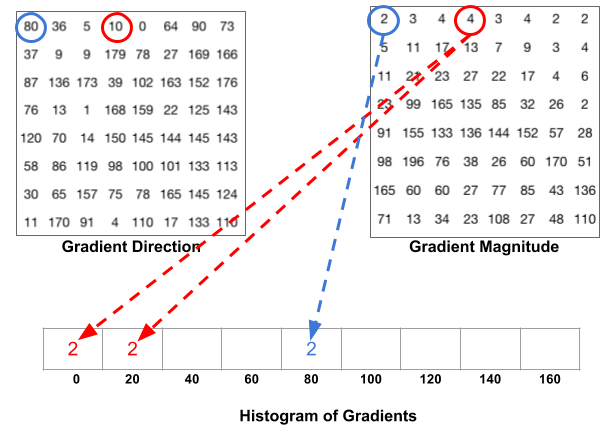
[](https://www.learnopencv.com/wp-content/uploads/2016/12/hog-cell-gradients.png)Center : The RGB patch and gradients represented using arrows. Right : The gradients in the same patch represented as numbers

If you are a beginner in computer vision, the image in the center is very informative. It shows the patch of the image overlaid with arrows showing the gradient — the arrow shows the direction of gradient and its length shows the magnitude. Notice how the direction of arrows points to the direction of change in intensity and the magnitude shows how big the difference is.

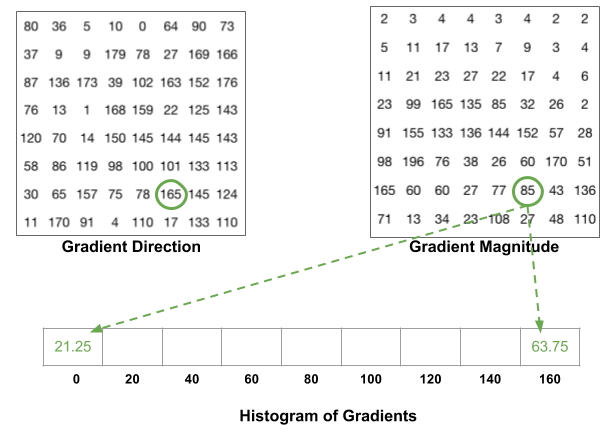
On the right, we see the raw numbers representing the gradients in the 8×8 cells with one minor difference — the angles are between 0 and 180 degrees instead of 0 to 360 degrees. These are called **“unsigned” gradients**because a gradient and it’s negative are represented by the same numbers. In other words, a gradient arrow and the one 180 degrees opposite to it are considered the same. But, why not use the 0 – 360 degrees ? Empirically it has been shown that unsigned gradients work better than signed gradients for pedestrian detection. Some implementations of HOG will allow you to specify if you want to use signed gradients.

The next step is to create a histogram of gradients in these 8×8 cells. The histogram contains 9 bins corresponding to angles 0, 20, 40 … 160.

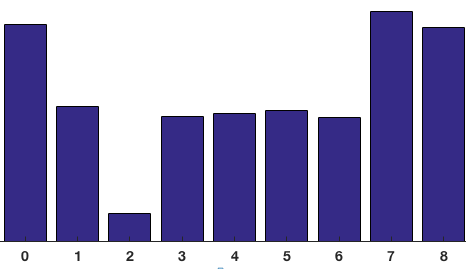
The following figure illustrates the process. We are looking at magnitude and direction of the gradient of the same 8×8 patch as in the previous figure. A bin is selected based on the direction, and the vote ( the value that goes into the bin ) is selected based on the magnitude. Let’s first focus on the pixel encircled in blue. It has an angle ( direction ) of 80 degrees and magnitude of 2. So it adds 2 to the 5th bin. The gradient at the pixel encircled using red has an angle of 10 degrees and magnitude of 4. Since 10 degrees is half way between 0 and 20, the vote by the pixel splits evenly into the two bins.

[](https://www.learnopencv.com/wp-content/uploads/2016/12/hog-histogram-1.png)

There is one more detail to be aware of. If the angle is greater than 160 degrees, it is between 160 and 180, and we know the angle wraps around making 0 and 180 equivalent. So in the example below, the pixel with angle 165 degrees contributes proportionally to the 0 degree bin and the 160 degree bin.

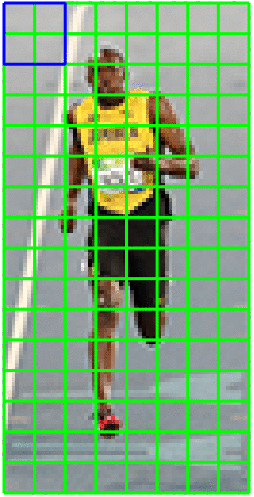
[](https://www.learnopencv.com/wp-content/uploads/2016/12/hog-histogram-2.png)

The contributions of all the pixels in the 8×8 cells are added up to create the 9-bin histogram. For the patch above, it looks like this

[](https://www.learnopencv.com/wp-content/uploads/2016/12/histogram-8x8-cell.png)

In our representation, the y-axis is 0 degrees. You can see the histogram has a lot of weight near 0 and 180 degrees, which is just another way of saying that in the patch gradients are pointing either up or down.

**Step 4 : 16×16 Block Normalization**

[](https://www.learnopencv.com/wp-content/uploads/2016/12/hog-16x16-block-normalization.gif)  
In the previous step, we created a histogram based on the gradient of the image. Gradients of an image are sensitive to overall lighting. If you make the image darker by dividing all pixel values by 2, the gradient magnitude will change by half, and therefore the histogram values will change by half. Ideally, we want our descriptor to be independent of lighting variations. In other words, we would like to “normalize” the histogram so they are not affected by lighting variations.

Before I explain how the histogram is normalized, let’s see how a vector of length 3 is normalized.

Let’s say we have an RGB color vector [ 128, 64, 32 ]. The length of this vector is \sqrt{128^2 + 64^2 + 32^2} = 146.64. This is also called the L2 norm of the vector. Dividing each element of this vector by 146.64 gives us a normalized vector [0.87, 0.43, 0.22]. Now consider another vector in which the elements are twice the value of the first vector 2 x [ 128, 64, 32 ] = [ 256, 128, 64 ]. You can work it out yourself to see that normalizing [ 256, 128, 64 ] will result in [0.87, 0.43, 0.22], which is the same as the normalized version of the original RGB vector. You can see that normalizing a vector removes the scale.

Now that we know how to normalize a vector, you may be tempted to think that while calculating HOG you can simply normalize the 9×1 histogram the same way we normalized the 3×1 vector above. It is not a bad idea, but a better idea is to normalize over a bigger sized block of 16×16. A 16×16 block has 4 histograms which can be concatenated to form a 36 x 1 element vector and it can be normalized just the way a 3×1 vector is normalized. The window is then moved by 8 pixels ( see animation ) and a normalized 36×1 vector is calculated over this window and the process is repeated.

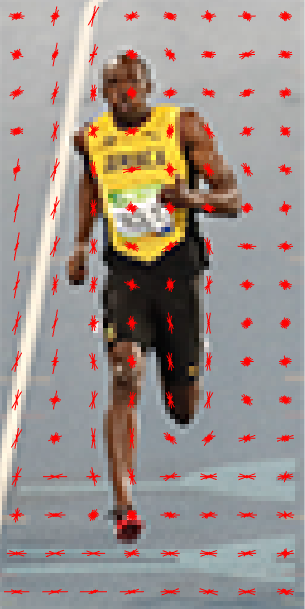
Step 5 : Calculate the HOG feature vector

To calculate the final feature vector for the entire image patch, the 36×1 vectors are concatenated into one giant vector. What is the size of this vector ? Let us calculate

How many positions of the 16×16 blocks do we have ? There are 7 horizontal and 15 vertical positions making a total of 7 x 15 = 105 positions.

Each 16×16 block is represented by a 36×1 vector. So when we concatenate them all into one gaint vector we obtain a 36×105 = **3780**dimensional vector.

Visualizing Histogram of Oriented Gradients

[](https://www.learnopencv.com/wp-content/uploads/2016/12/hog-visualization.png)

The HOG descriptor of an image patch is usually visualized by plotting the 9×1 normalized histograms in the 8×8 cells. See image on the side. You will notice that dominant direction of the histogram captures the shape of the person, especially around the torso and legs.

参考资料：

<https://www.learnopencv.com/histogram-of-oriented-gradients/>

<https://blog.csdn.net/dss_dssssd/article/details/82663779>

<https://www.cnblogs.com/zhehan54/p/6723956.html>

<https://blog.csdn.net/coming_is_winter/article/details/72850511>

1. 查找资料理解hog方式进行人脸识别

This is a widely used face detection model, based on HoG features and SVM. You can read more about HoG in [our post](https://www.learnopencv.com/histogram-of-oriented-gradients/). The model is built out of 5 HOG filters – front looking, left looking, right looking, front looking but rotated left, and a front looking but rotated right. The model comes embedded in the [header file](https://github.com/davisking/dlib/blob/master/dlib/image_processing/frontal_face_detector.h) itself.

The dataset used for training, consists of 2825 images which are obtained from LFW dataset and manually annotated by Davis King, the author of Dlib. It can be downloaded from [here](http://dlib.net/files/data/dlib_face_detector_training_data.tar.gz).

**Pros**

Fastest method on CPU

Works very well for frontal and slightly non-frontal faces

Light-weight model as compared to the other three.

Works under small occlusion

Basically, this method works under most cases except a few as discussed below.

**Cons**

The major drawback is that it does not detect small faces as it is trained for minimum face size of 80×80. Thus, you need to make sure that the face size should be more than that in your application. You can however, train your own face detector for smaller sized faces.

The bounding box often excludes part of forehead and even part of chin sometimes.

Does not work very well under substantial occlusion

Does not work for side face and extreme non-frontal faces, like looking down or up.

参考资料：

<https://www.learnopencv.com/face-detection-opencv-dlib-and-deep-learning-c-python/>

# 任务7 编程识别陌生人

## 任务描述

**内 容**：训练人脸识别模型，并识别陌生人。

**学 时**：2

**知识点**：使用dlib库训练人脸识别模型、识别陌生人

**重点**：使用dlib库训练人脸识别模型、识别陌生人

**难点**：使用dlib库训练人脸识别模型、识别陌生人

## 授课思路

本任务用到了人脸识别这个新技术，需提供给同学们完整的代码以帮助他们快速掌握知识。

本次任务在任务实现中提供‘训练人脸识别模型’的完整代码，提供‘识别陌生人’的完整代码。学生通过代码了解并熟悉人脸识别。

## 任务指导

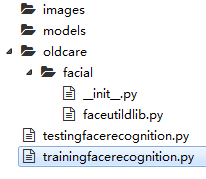
* 1. 使用上个任务中的人脸图像采集程序采集2~3人的人脸图像；
  2. 使用dlib库训练人脸识别模型；
  3. 实时识别陌生人。

## 任务实现

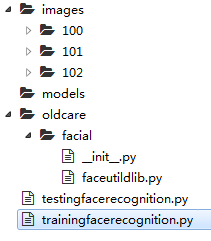
本次子任务的完整代码放在 **任务源代码/任务7.编程识别陌生人** 中。

1. 使用上个任务中的人脸图像采集程序采集2~3人的人脸图像

本次子任务的目录结构为:



将图像存放在images目录下。最终效果如下图所示：



1. 使用dlib库训练人脸识别模型

打开faceutildlib.py，在类中新添加2个方法，一个方法用于人脸识别，一个方法用于训练人脸识别模型。添加后的完整代码如下：

# -\*- coding: utf-8 -\*-

'''

使用dlib实现人脸检测

'''

import face\_recognition

import cv2

import pickle

import os

class FaceUtil:

# 超参数

detection\_method = 'hog' # either 'hog' or 'cnn'. default is hog.

tolerance = 0.3

def \_\_init\_\_(self, encoding\_file\_path = None):

if encoding\_file\_path:

self.load\_embeddings(encoding\_file\_path)

# load embeddings

def load\_embeddings(self, encoding\_file\_path):

# load the known faces and embeddings

print("[INFO] loading face encodings...")

self.data = pickle.loads(open(encoding\_file\_path,"rb").read())

# face detection

def get\_face\_location(self, image):

face\_location\_list = []

gray = cv2.cvtColor(image, cv2.COLOR\_BGR2GRAY)

face\_locations = face\_recognition.face\_locations(

gray,number\_of\_times\_to\_upsample=1,

model = self.detection\_method)

# 人脸位置

for (top, right, bottom, left) in face\_locations:

face\_location\_list.append((left, top, right, bottom))

return face\_location\_list

# face recognition

def get\_face\_location\_and\_name(self,image):

# convert the input frame from BGR to RGB

rgb = cv2.cvtColor(image, cv2.COLOR\_BGR2RGB)

# detect the (x, y)-coordinates of the bounding boxes

#corresponding to each face in the input frame, then

#compute the facial embeddings for each face

boxes = face\_recognition.face\_locations(

rgb, model = self.detection\_method)

encodings = face\_recognition.face\_encodings(rgb, boxes)

# initialize the list of names for each face detected

names = []

# loop over the facial embeddings

for encoding in encodings:

# attempt to match each face in the input image to

#our known encodings

matches = face\_recognition.compare\_faces(

self.data["encodings"], encoding,

tolerance = self.tolerance)

name = "Unknown"

# check to see if we have found a match

if True in matches:

# find the indexes of all matched faces then

# initialize a dictionary to count the total number

# of times each face was matched

matched\_idxs=[i for (i, b) in enumerate(matches) if b]

counts = {}

# loop over the matched indexes and maintain a count

# for each recognized face face

for i in matched\_idxs:

name = self.data["names"][i]

counts[name] = counts.get(name, 0) + 1

# determine the recognized face with the largest

# number of votes (note: in the event of an unlikely

# tie Python will select first entry in the

#dictionary)

name = max(counts, key=counts.get)

# update the list of names

names.append(name)

face\_location\_list = []

for ((top, right, bottom, left)) in boxes:

face\_location\_list.append((left, top, right, bottom))

return face\_location\_list, names

def save\_embeddings(self, image\_paths, output\_encoding\_file\_path):

warning = ''

# initialize the list of known encodings and known names

known\_encodings = []

known\_names = []

# loop over the image paths

for (i, image\_path) in enumerate(image\_paths):

# extract the person name from the image path

print("[INFO] processing image {}/{}"

.format(i + 1, len(image\_paths)))

name = image\_path.split(os.path.sep)[-2] # person name

# load the input image and convert it from

#RGB (OpenCV ordering) to dlib ordering (RGB)

image = cv2.imread(image\_path)

rgb = cv2.cvtColor(image, cv2.COLOR\_BGR2RGB)

# detect the (x, y)-coordinates of the bounding boxes

#corresponding to each face in the input image

boxes = face\_recognition.face\_locations(

rgb, model = self.detection\_method)

# compute the facial embedding for the face

encodings = face\_recognition.face\_encodings(rgb, boxes)

if len(encodings) != 1:

os.remove(image\_path)

warning += '[WARNING] detected %d faces in %s.'

warning += ' This file is deleted.\n' %(

len(encodings), image\_path)

continue

# loop over the encodings

for encoding in encodings:

# add each encoding + name to our set of known names

#and encodings

known\_encodings.append(encoding)

known\_names.append(name)

# dump the facial encodings + names to disk

print("[INFO] serializing encodings...")

data = {"encodings": known\_encodings, "names": known\_names}

f = open(output\_encoding\_file\_path, "wb")

f.write(pickle.dumps(data))

f.close()

if warning:

print(warning)

其中**get\_face\_location\_and\_name()**方法用于人脸识别，得到人脸的位置和对应的名字。

其中**save\_embeddings()**方法用于训练人脸识别模型，并把模型保存到硬盘中。

有了训练人脸识别模型的方法，现调用这个方法训练一下模型。

在根目录下新建一个模块，命名为trainingfacerecognition.py。完整代码如下：

# -\*- coding: utf-8 -\*-

'''

训练人脸识别模型

'''

# import the necessary packages

from imutils import paths

from oldcare.facial import FaceUtil

# global variable

dataset\_path = 'images'

output\_encoding\_file\_path = 'models/face\_recognition\_hog.pickle'

# grab the paths to the input images in our dataset

print("[INFO] quantifying faces...")

image\_paths = list(paths.list\_images(dataset\_path))

if len(image\_paths) == 0:

print('[ERROR] no images to train.')

else:

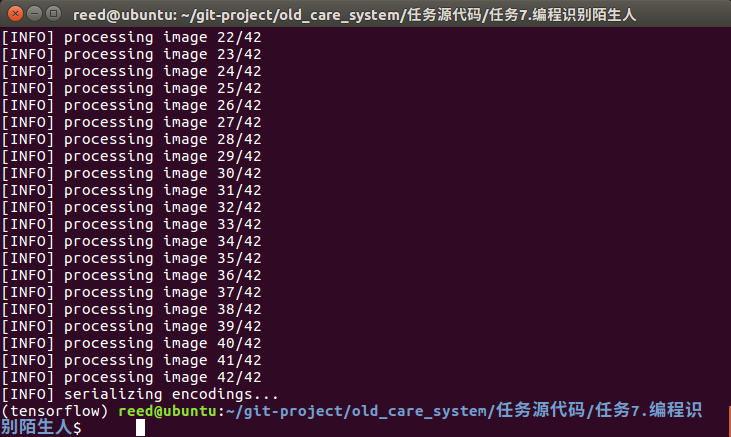
faceutil = FaceUtil()

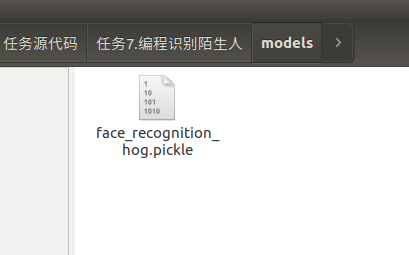
print("[INFO] training face embeddings...")

faceutil.save\_embeddings(image\_paths, output\_encoding\_file\_path)

执行trainingfacerecognition.py会训练模型，并把模型文件保存到硬盘中。保存路径为**models/face\_recognition\_hog.pickle**。

程序运行后结果如下：





1. 实时识别陌生人

模型已经训练并保存在硬盘中，下一步我们需要加载这个模型并使用它做人脸识别。我们在实例化FaceUtil这个类的时候，需要把构造函数的那个参数传进去，那是我们训练的模型文件的路径。

在根目录下新建一个module，命名为**testingfacerecognition.py**。添加如下代码：

# -\*- coding: utf-8 -\*-

'''

测试人脸识别模型

用法：

python testingfacerecognition.py

python testingfacerecognition.py --filename room\_01.mp4

'''

# import the necessary packages

from oldcare.facial import FaceUtil

import imutils

import cv2

import time

import argparse

# 传入参数

ap = argparse.ArgumentParser()

ap.add\_argument("-f", "--filename", required=False, default = '',

help="")

args = vars(ap.parse\_args())

# 全局变量

facial\_recognition\_model\_path = 'models/face\_recognition\_hog.pickle'

input\_video = args['filename']

# 初始化摄像头

if not input\_video:

vs = cv2.VideoCapture(0)

time.sleep(2)

else:

vs = cv2.VideoCapture(input\_video)

# 初始化人脸识别模型

faceutil = FaceUtil(facial\_recognition\_model\_path)

# 不断循环

while True:

# grab the current frame

(grabbed, frame) = vs.read()

# if we are viewing a video and we did not grab a frame, then we

# have reached the end of the video

if input\_video and not grabbed:

break

if not input\_video:

frame = cv2.flip(frame, 1)

# resize the frame, convert it to grayscale, and then clone the

# original frame so we can draw on it later in the program

frame = imutils.resize(frame, width = 600)

face\_location\_list, names = faceutil.get\_face\_location\_and\_name(

frame)

# loop over the face bounding boxes

for ((left, top, right, bottom), name) in zip(

face\_location\_list,

names):

# display label and bounding box rectangle on the output frame

cv2.putText(frame, name, (left, top - 10),

cv2.FONT\_HERSHEY\_SIMPLEX, 0.5, (0, 0, 255), 2)

cv2.rectangle(frame, (left, top), (right, bottom),

(0, 0, 255), 2)

# show our detected faces along with smiling/not smiling labels

cv2.imshow("Face Recognition", frame)

# Press 'ESC' for exiting video

k = cv2.waitKey(1) & 0xff

if k == 27:

break

# cleanup the camera and close any open windows

vs.release()

cv2.destroyAllWindows()

执行python testingfacerecognition.py即可运行人脸识别程序。此程序可用于识别陌生人。程序运行结果如下图所示：

