运用BP神经网络实现性别检测（工作报告）

标签: OpenCV BP神经网 人脸识别 性别识别 python

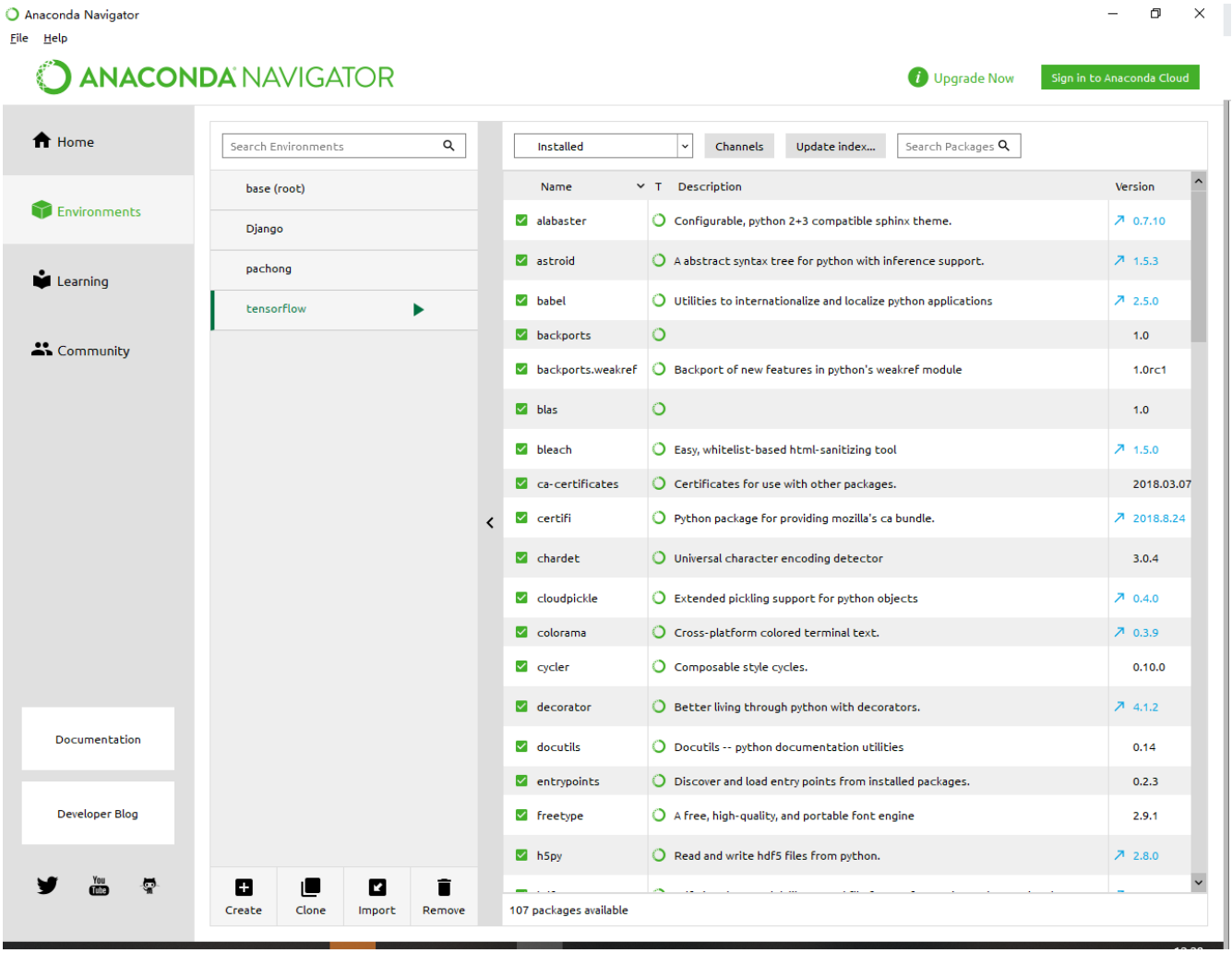
1. 兴趣的起源

之前去超市的时候，发现大润发里面有一个显示屏，人走过去的时候就会检测人脸，并对性别和年龄做出判断。我对此十分感兴趣，在上了机器学习这门课后，听老师说实现人脸识别不是一件困难的事情，这就激发了我自己动手实现人脸识别和性别识别的梦想。

2. 准备工作

由于完全是一个小白进行开发，中间走了很多弯路，所以在这里重新梳理了一下所用到的工具，并对其用法进行一个简要的介绍。

anaconda

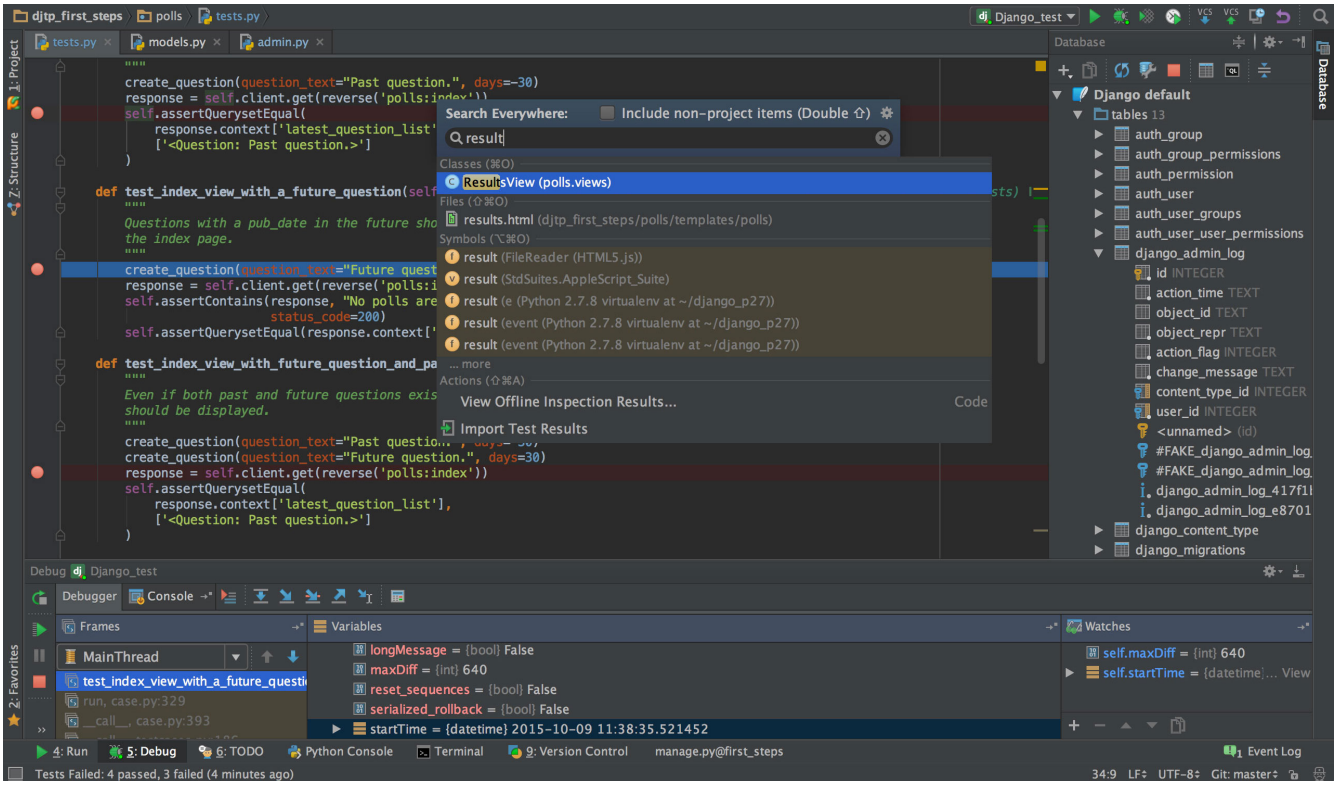


anaconda界面

anaconda是一个专业的python管理器。之前我直接在官网安装python，使用pip进行管理，到后面发现很多包pip根本安装不了。anaconda的优点在于：

1. anaconda提供了界面化的设置。能够十分方便的下载和删除应用包。还能很容易的查看已安装的包和能安装的包。
2. anaconda能够十分方便地对Python环境进行设置。能够很快地独立创建一个新环境，十分方便管理。

pycharm

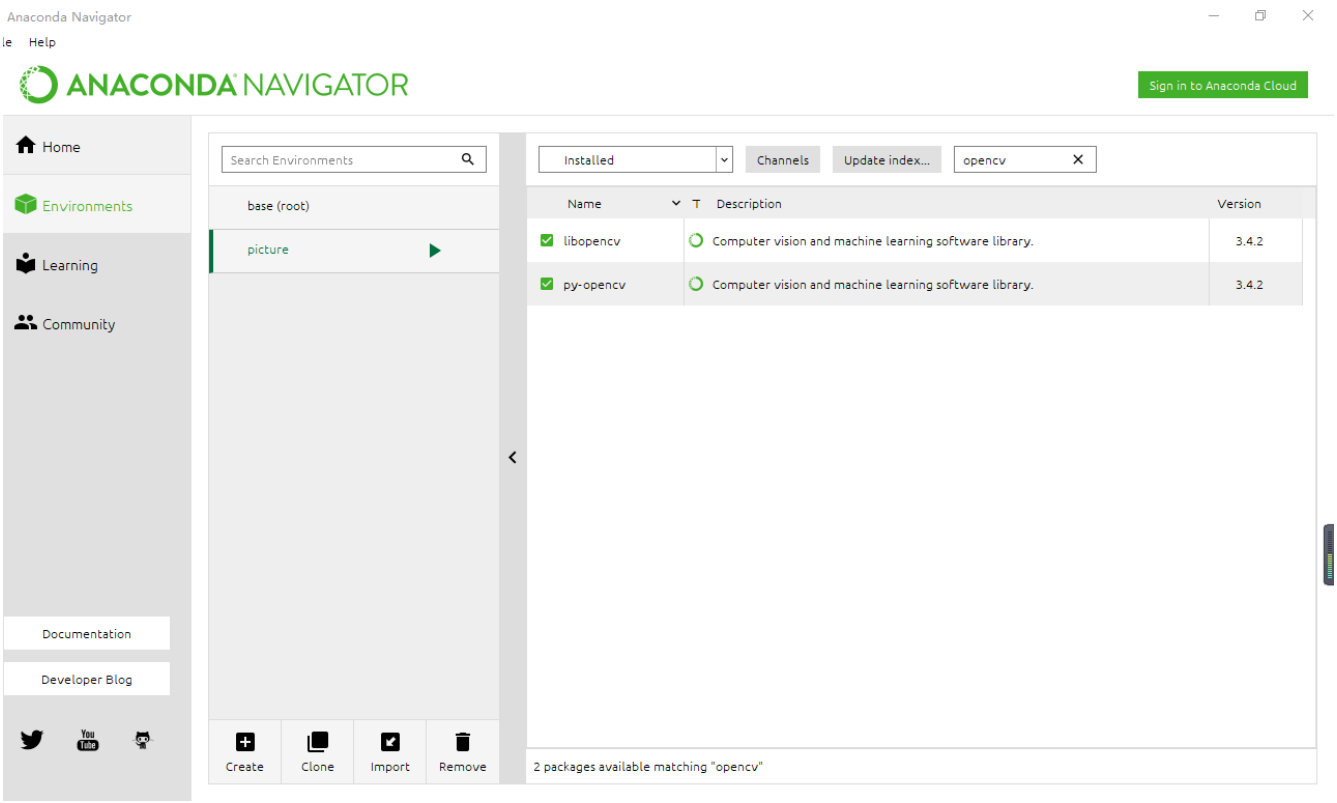


pycharm应用界面

pycharm是一个python编程的IDE，安装十分便捷。同时里面也支持anaconda环境。

OpenCV

OpenCV是一个基于BSD许可（开源）发行的跨平台计算机视觉库，可以运行在Linux、Windows、Android和Mac OS操作系统上。它轻量级而且高效——由一系列 C 函数和少量 C++ 类构成，同时提供了Python、Ruby、MATLAB等语言的接口，实现了图像处理和计算机视觉方面的很多通用算法。 我们可以使用OpenCV的python接口，在anaconda里面可以搜索opencv安装相应的包。



anaconda下搜索opencv

3. 简单的尝试

在下载好python编辑器并且配置好环境之后，我们就可以开始大胆尝试了。 所有的源码都在我GitHub仓库中

3.1 人脸检测

第一步我们需要检测出人脸，这里我们利用opencv自带的分类器识别人脸，并画出框框显示人脸区域。先展示代码然后再进行说明。

```
import cv2

filename =  'C:/Users/zhao1/Desktop/show/test/example/1.jpg'
casvade_face_name='C:/Users/zhao1/Desktop/show/test/cascades/haarcascade_frontalface_default.xml'
def detect(filename) :
    face_cascade=cv2.CascadeClassifier(casvade_face_name)
    img=cv2.imread(filename)
    gray=cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)

    faces=face_cascade.detectMultiScale(gray,1.5,3)

    for (x,y,w,h) in faces:
        img = cv2.rectangle(img,(x,y),(x+w,y+h),(255,0,0),2)
    cv2.namedWindow('find')
    cv2.imshow('face',img)
    cv2.imwrite('C:/Users/zhao1/Desktop/show/test/人脸.jpg ',img)
    cv2.waitKey(0)
```

```
detect(filename)
```

> import cv2

这句话是导入opencv应用包，我们需要确定自己的环境中已经下载并安装了opencv.

> filename = 'C:/Users/zhao1/Desktop/show/test/example/1.jpg'
> casvade_face_name='C:/Users/zhao1/Desktop/show/test/cascades/haarcascade_frontalface_default.xml'

filename是要检测的照片路径，casvade_face_name是人脸分类器的绝对路径。这里我使用的是绝对路径，如果想要工程的可移植性更好的话最好用相对路径。

注意：在python中，路径要小心反斜杠产生转义字符。所以这里使用‘/’代替原本路径的‘\’，具体可以学习一下python语法

> face_cascade=cv2.CascadeClassifier(casvade_face_name) img=cv2.imread(filename)

第一行是读取分类器，并返回一个值；第二行是读取图片文件。

> gray=cv2.cvtColor(img,cv2.COLOR_BGR2GRAY) faces=face_cascade.detectMultiScale(gray,1.5,3)

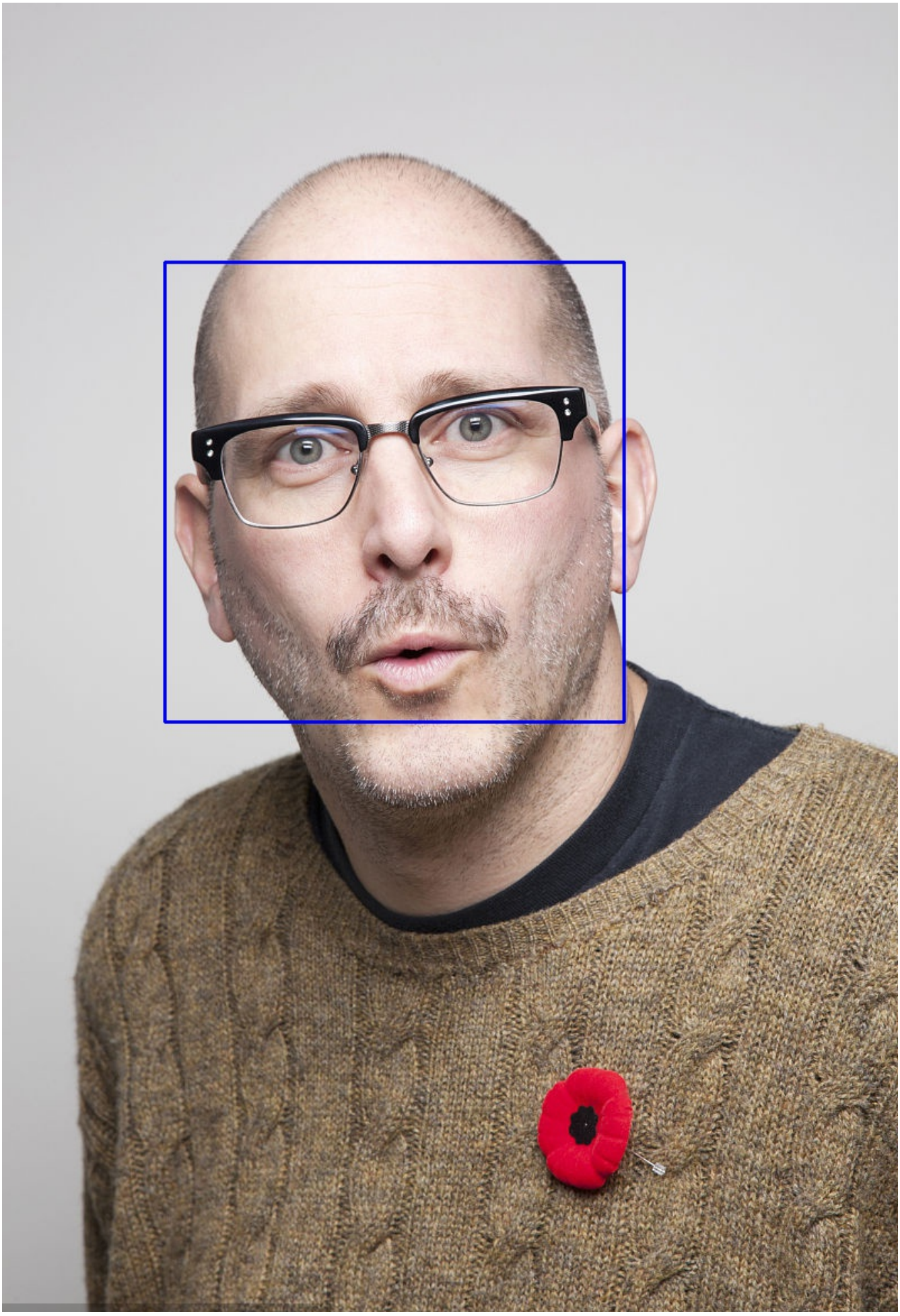
第一行是将读到的图片进行灰度化处理，这是因为opencv处理是对灰度照片进行处理。第二行detectMultiScale是对人脸检测的函数，后面的参数可以点进去查看详情

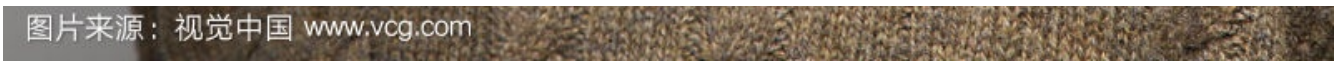
```
def detectMultiScale(self, image, scaleFactor=None, minNeighbors=None, flags=None,
minSize=None, maxSize=None): # real signature unknown; restored from __doc__
        """
        detectMultiScale(image[, scaleFactor[, minNeighbors[, flags[, minSize[,
maxSize]]]]]) -> objects
        .   @brief Detects objects of different sizes in the input image. The detected
objects are returned as a list
        .   of rectangles.
        .
        .   @param image Matrix of the type CV_8U containing an image where objects are
detected.
        .   @param objects Vector of rectangles where each rectangle contains the
detected object, the
        .   rectangles may be partially outside the original image.
        .   @param scaleFactor Parameter specifying how much the image size is reduced at
each image scale.
        .   @param minNeighbors Parameter specifying how many neighbors each candidate
rectangle should have
        .   to retain it.
        .   @param flags Parameter with the same meaning for an old cascade as in the
function
        .   cvHaarDetectObjects. It is not used for a new cascade.
        .   @param minSize Minimum possible object size. Objects smaller than that are
ignored.
        .   @param maxSize Maximum possible object size. Objects larger than that are
ignored. If `maxSize == minSize` model is evaluated on single scale.
        .
        .   The function is parallelized with the TBB library.
```

这是一个经常使用的函数。

```
    for (x,y,w,h) in faces: img = cv2.rectangle(img,(x,y),(x+w,y+h),(255,0,0),2) cv2.namedWindow('find')
cv2.imshow('face',img) cv2.imwrite('C:/Users/zhao1/Desktop/show/test/人脸.jpg ',img)
```

这些是画出方框并保存。其中(x,y,w,h)里面的x,y是指检测到的人脸左上角的坐标，w,h分别是区域的长度和宽度。 下面是这个例程最后的效果，大家可以添加自己的图片。





人脸识别效果图

上面我们就实现了对人脸的简单识别。

3.2 在视频中检测人脸

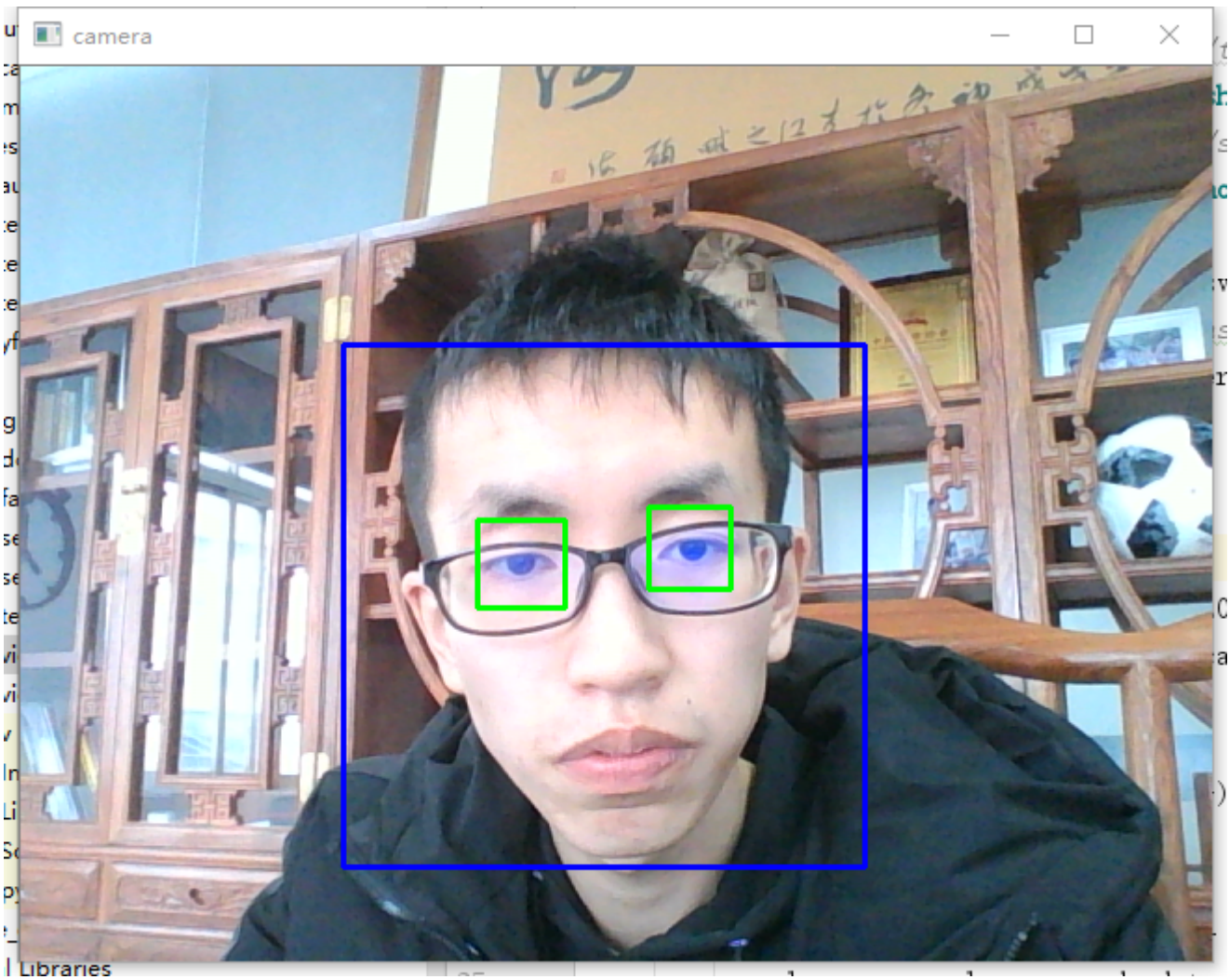
在实现图片中检测人脸之后，我们尝试从视频中实现人脸的识别。主体代码和之前的类似，只是增加了摄像头的驱动还有图像的获取。下面具体说明。

```
camera= cv2.VideoCapture(0)
```

要使用摄像头，需要使用cv2.VideoCapture(0)创建VideoCapture对象，参数：0指的是摄像头的编号。如果你电脑上有两个摄像头的话，访问第2个摄像头就可以传入1。

```
ret, frame=camera.read()
```

这是从摄像头中读取图片，frame相当与前面的img，后面的处理与前面类似。下面是效果图。



视频实现人脸和眼睛识别 在这里我还加入了眼睛的识别，感兴趣的可以查看源代码。

4. 进阶应用

在掌握简单的应用之后我开始了进阶之路。 在这里我们需要应用BP神经网络对样本训练之后再判断视频中人的性别。 这个源代码是我经过认真整理之后的代码，里面的注释写的很详细，大家可以仔细阅读。

4.1 BP神经网络

后面的代码都是根据老师的课件进行分析的。 人工神经网络(ANNs)提供了一种普遍而且实用的方法，来从样例中学习值为实数、离散或向量的函数。 反向传播算法使用梯度下降来调节网络参数以最佳拟合由输入-输出对组成的训练集合。 我们结合代码和资料学习会对此理解更深刻。

4.2 核心函数学习

下面先列出算法的核心函数，后面再对照老师的资料详细讲解

```
'''
    训练样本：百度的图片。女生标签为0，男生标签为1.
    训练方法：简单的梯度下降法
    参考：https://blog.csdn.net/yunyunyx/article/details/80539222
'''

'''
设置一个隐藏层，为每张照片像素值-->pixel_mat     隐藏层神经元个数-->1
输入为每张图片的灰度像素矩阵
x_train:训练样本的像素数据
y_train：训练样本的标签
w：输出层权重
b：输出层偏置
w_h：隐藏层权重
b_h：隐藏层偏置
step：循环步数
'''
def mytrain(x_train,y_train):

    step=int(input('mytrain迭代步数：'))
    a=double(input('学习因子：'))
    inn = pixel_mat                                    #输入神经元个数
    hid = int(input('隐藏层神经元个数：'))             #隐藏层神经元个数
    out = 1                                            #输出层神经元个数

    w = np.random.randn(out,hid)
    w = np.mat(w)
    b = np.mat(np.random.randn(out,1))
    w_h = np.random.randn(hid,inn)
    w_h = np.mat(w_h)
    b_h = np.mat(np.random.randn(hid,1))

    for i in range(step):
        #打乱训练样本
        r=np.random.permutation(photonum)
        x_train = x_train[:,r]
        y_train = y_train[:,r]
        for j in range(photonum):
            x = np.mat(x_train[:,j])
```

```
            x = x.reshape((pixel_mat,1))
            y = np.mat(y_train[:,j])
            y = y.reshape((1,1))
            hid_put = layerout(w_h,b_h,x)
            out_put = layerout(w,b,hid_put)

            #更新公式的实现
            o_update = np.multiply(np.multiply((y-out_put),out_put),(1-out_put))
               #计算输出单元误差项，y->tk
            h_update = np.multiply(np.multiply(np.dot((w.T),np.mat(o_update)),hid_put),
(1-hid_put))      #隐藏单元误差项

            outw_update = a*np.dot(o_update,(hid_put.T))
               #从隐藏层到输出层的dw
            outb_update = a*o_update
            hidw_update = a*np.dot(h_update,(x.T))
            hidb_update = a*h_update

            w = w + outw_update
              #更新参数
            b = b+ outb_update
            w_h = w_h +hidw_update
            b_h =b_h +hidb_update

    return w,b,w_h,b_h
```

这个函数输入有两个矩阵，一个是样本数据，一个是样本标签，至于如何得到这些数据，我们后面再说，现在只要关注函数本身就好。这里的样本数据是像素值，样本标签是人为设定的，我们设置0为女生，1为男生。

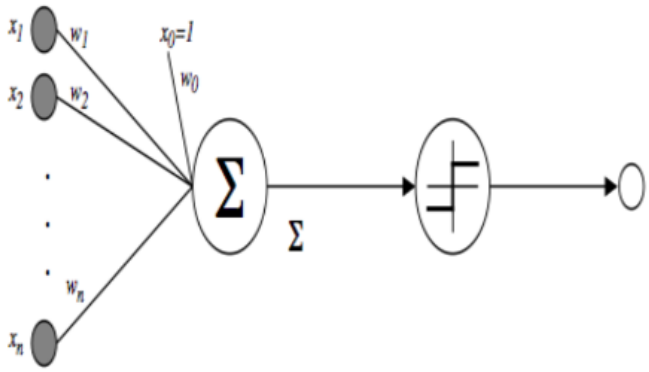
首先，函数要求我们手动输入几个参数，分别是迭代步数、学习因子和隐藏神经元个数。迭代步数会影响我们最后的准确度和学习所需要的时间，学习因子是反向传播算法的一个参数，而隐藏神经元个数和输入的像素个数有关。

接下来就是对参数的初始化。

> w：输出层权重 b：输出层偏置 w_h：隐藏层权重 b_h：隐藏层偏置 step：循环步数 w = np.random.randn(out,hid) w = np.mat(w) b = np.mat(np.random.randn(out,1)) w_h = np.random.randn(hid,inn) w_h = np.mat(w_h) b_h = np.mat(np.random.randn(hid,1))

每个参数上面都有注释，现在说明一下参数 和老师PDF里面的对应情况。 我们总共有三层，输入层，隐藏层和输出层。输入层神经元个数是照片像素个数，输出是神经元只有一个，隐藏层个数人为设定。 看到关于感知器的那页。

感知器



$$o(x_1,\ldots,x_n)=\begin{cases}1 & \text{if } w_0+w_1x_1+\cdots+w_nx_n>0\\ -1 & \text{otherwise.}\end{cases}$$

简化表示:

$$o(\vec{x})=\begin{cases}1 & \text{if } \vec{w}\cdot\vec{x}>0\\ -1 & \text{otherwise.}\end{cases}$$

w:对应就是由$w_1,w_2,\ldots w_n$组成的向量，只不过是输出层的权重。同理w_h也是如此。 b:输出层的偏置，对应的就是w_0。b_h也是如此。

有了输出层和隐藏层，我们就要处理输入层，下面这段就是对输入层的处理。

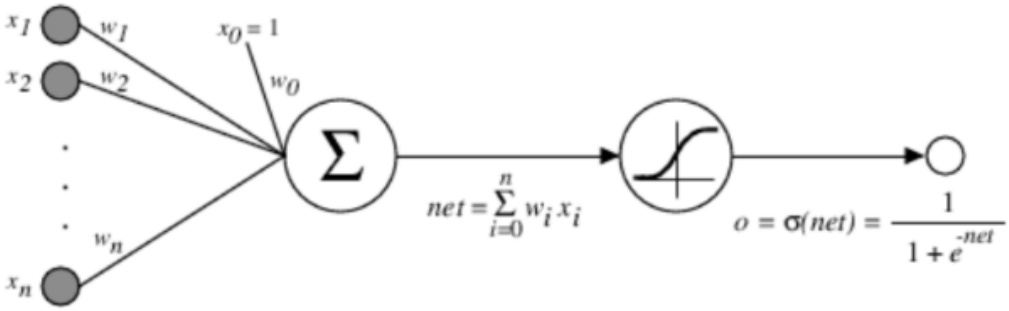
打乱训练样本

```
r=np.random.permutation(photonum)
x_train = x_train[:,r]
y_train = y_train[:,r]
for j in range(photonum):
    x = np.mat(x_train[:,j])
    x = x.reshape((pixel_mat,1))
    y = np.mat(y_train[:,j])
    y = y.reshape((1,1))
```

还是上面的对照表，实际上我们输入的x_train是一个很大的矩阵，包含了所有照片的像素值，每一列就代表一张照片的像素向量。所以x_train[:,r]代表的就是第r张照片的像素向量。单独整理成矩阵之后，x对应的就是$x_1,x_2,\ldots x_n$这些输入。**值得注意的是，偏置也有输入x_0,只不过默认值为1，所以在PDF中没有表示出来。**

有了数据的初始化之后，我们就要用sigmoid函数对结果进行初始化，以便于后面的更新。看到PDF有关于sigmoid单元的描述

Sigmoid 单元



$$\sigma(x) = \frac{1}{1 + e^{-x}}$$

$$\frac{d\sigma(x)}{dx} = \sigma(x)(1 - \sigma(x))$$

可得梯度下降法则用于训练：

- 单个 sigmoid 单元
- 由 sigmoid 单元构成的多层网络 $\rightarrow$ 反向传播（Backpropagation）

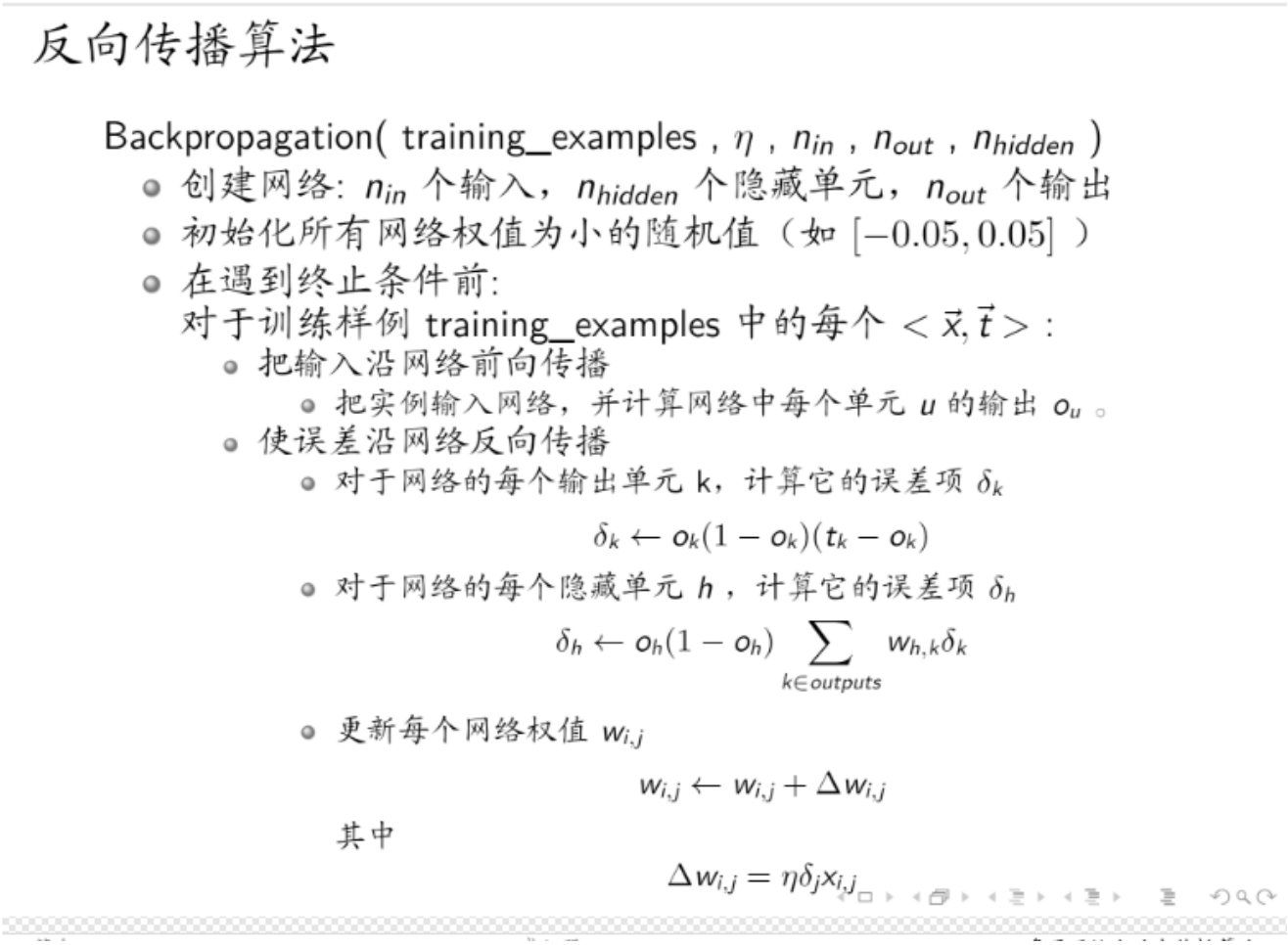
这里使用了一个layerout函数，是自己写的

```
hid_put = layerout(w_h,b_h,x) out_put = layerout(w,b,hid_put)
```

这里要详细看一下图中的那个求和了。$net = \sum_{i=0}^{n} w_i x_i = w_0 x_0 + w_1 x_1 + \ldots w_n x_n$ 由于$x_0 = 1$,所以上式也可以写作$net = \sum_{i=0}^{n} w_i x_i = w_0 + w_1 x_1 + \ldots w_n x_n$ 因此在这个layerout函数中，b_h代表的就是w_0,w_h和x的向量积就是后面的累加。这两句话的就是先算出输入层到隐藏层的值，再将这个输出值作为输入，计算隐藏层到输出的值。

经过以此计算之后，我们需要根据结果更新误差并且更新参数。我们可以查看PDF中关于反向传播算法的介绍。

简介 感知器 多层网络和反向传播算法



对应这部分的代码是代码中 *更新公式的实现* 后面那一部分，我们将一句一句介绍。

```
    o_update = np.multiply(np.multiply((y-out_put),out_put),(1-out_put)) #计算输出单元误差项，
```

这是计算输出单元误差，其中multipiy是python中的一个计算乘积的方法。out_put对应的o_k，而y则对应的是t_k。o_update对应的是δ_k

```
    h_update = np.multiply(np.multiply(np.dot((w.T),np.mat(o_update)),hid_put),(1-hid_put)) #隐藏单元误差项
```

这里是计算隐藏单元的误差，h_updata对应的是δ_h，dot是python计算向量点乘的方法。w.T是将w矩阵转置，hid_put对应的是o_h。

之所以叫误差沿网络反向传播，是因为我们先算出输出层的误差，将这个误差作为输入，再计算隐藏层的误差。

计算完误差之后我们就需要跟新每个网络的权值了。

```
    outw_update = anp.dot(o_update,(hid_put.T)) #从隐藏层到输出层的dw outb_update = ao_update
hidw_update = anp.dot(h_update,(x.T)) hidb_update = ah_update
```

这里就是计算输入层到隐藏层，隐藏层到输出层的$\Delta w_{i,j}$，其中a对应的就是学习因子η。

后面几行就是更新权重了，这里不做过多解释。

4.3 数据的获得和处理

在学习完核心代码之后，我们只要获得输入的矩阵就可以啦。 其他的代码比较简单，注释因该能看懂，我就简略地介绍。 之前说过，输入矩阵是一个很大的矩阵，矩阵中的每一列就代表一张照片，比如我们由200张照片，每张照片由10*10=100个像素点，那么这个矩阵就是一个100*200的矩阵。 要得到这些数据，我们第一步需要找到人脸识别库，有很多开源的人脸识别库大家可以从里面获得各种各样想要的照片。 获得了照片之后，我们首先要识别人脸，这个在函数

```
def get_face_from_photo(i,path,spath):
```

这里将人脸识别出来之后灰度化再重新保存成带有编号的照片。 由于识别出来的人脸像素值太大，而且不同的照片包含的像素值不同，所以我们需要统一每张照片的像素值，用到了这个函数

```
def change_photo_size(path,spath):
```

这里统一像素值之后吧照片又保存到另外一个文件夹。

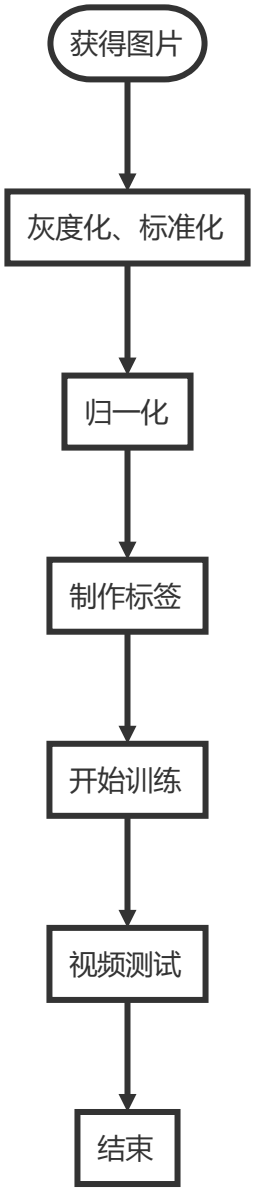
得到像素值统一的灰度图片之后，我们需要将所有图片都弄到一个大矩阵里面作为核心函数的输入，因此我们有这个函数

```
def read_photo_for_train(k,photo_path):
```

这个函数返回了一个矩阵。 这样我们就完成了整个数据处理啦！

4.4 工作流程

整个工作流程是从工程中主函数开始的。下面是流程图。



训练结束的时候相当于就是确定了每一层的权重，然后将待测的照片输入就可以。 获取视频中的人脸并进行处理的速度会有点慢，所以视频会比较卡。

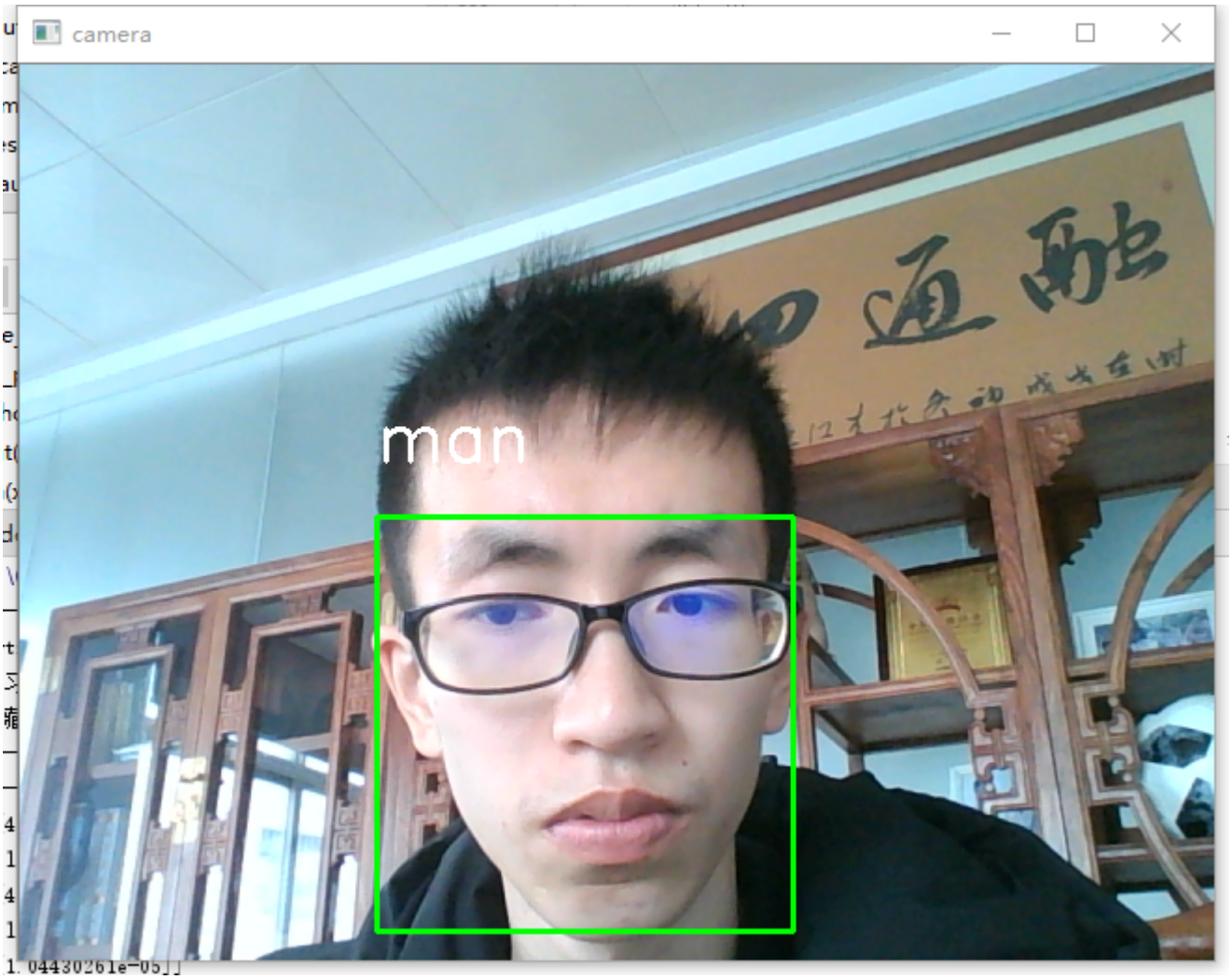
4.5 结果演示

下面我们来看一下演示的结果 运行程序之后会首先读取照片训练样本，当所有的照片处理完之后，会提醒你输入迭代步数，学习因子和隐藏层数。

```
video_sex_rec
C:\G\ProgramData\Anaconda3\envs\picture\python.exe C:/Users/zhao1/Desktop/show/test/program/video_sex_rec.py
————————————开始训练————————————————————————
mytrain迭代步数: 800
学习因子: 0.28
隐藏层神经元个数: 800
————————————训练结束————————————————————————
————————————视频测试————————————————————————
```

输入参数

注意我们输入的迭代步数和神经元个数都比较大，计算时间可能会比较久，测试的时候可以改成300左右



性别识别结果

5.总结

经过这一阵子的折腾，我深刻认识到想要学到真本领，还是需要自己亲自动手实践，在实践中才能将理论理解的更加深刻！