OpenCV

简介

OpenCV是一个基于BSD许可开源发行的跨平台计算机视觉库。拥有C++，Python和Java接口，并且支持Windows, Linux, Mac OS, iOS 和 Android系统。实现了图像处理和计算机视觉方面的很多通用算法。

模块	功能
Core	核心基础模块，定义了被所有其他模块和基本数据结构(包括重要的多维数组Mat)使用的基本函数、底层数据结构和算法函数
Imgproc	图像处理模块，包括：滤波、高斯模糊、形态学处理、几何变换、颜色空间转换及直方图计算等
Highgui	高层用户交互模块，包括：GUI、图像与视频I\O等
Video	视频分析,，运动分析及目标跟踪。
Calib3d	3D模块，包括：摄像机标定、立体匹配、3D重建等
Features2d	二维特征检测与描述模块，包括：图像特征检测、描述、匹配等
Objdetect	目标检测模块，如：人脸检测等
MI	机器学习模块，包括：支持向量机、神经网络等
Flann	最近邻开源库。包含一系列查找算法，自动选取最快算法的机制。
Imgcodecs	图像编解码模块，图像文件的读写操作
Photo	图像计算(处理)模块，图像修复及去噪。
Shape	形状匹配算法模块。描述形状、比较形状
Stitching	图像拼接
Superres	超分辨率模块
Videoio	视频读写模块，视频文件包括摄像头的输入。
Videostab	解决拍摄的视频稳定
Dnn	深度神经网络
contrib	可以引入额外模块

人脸定位

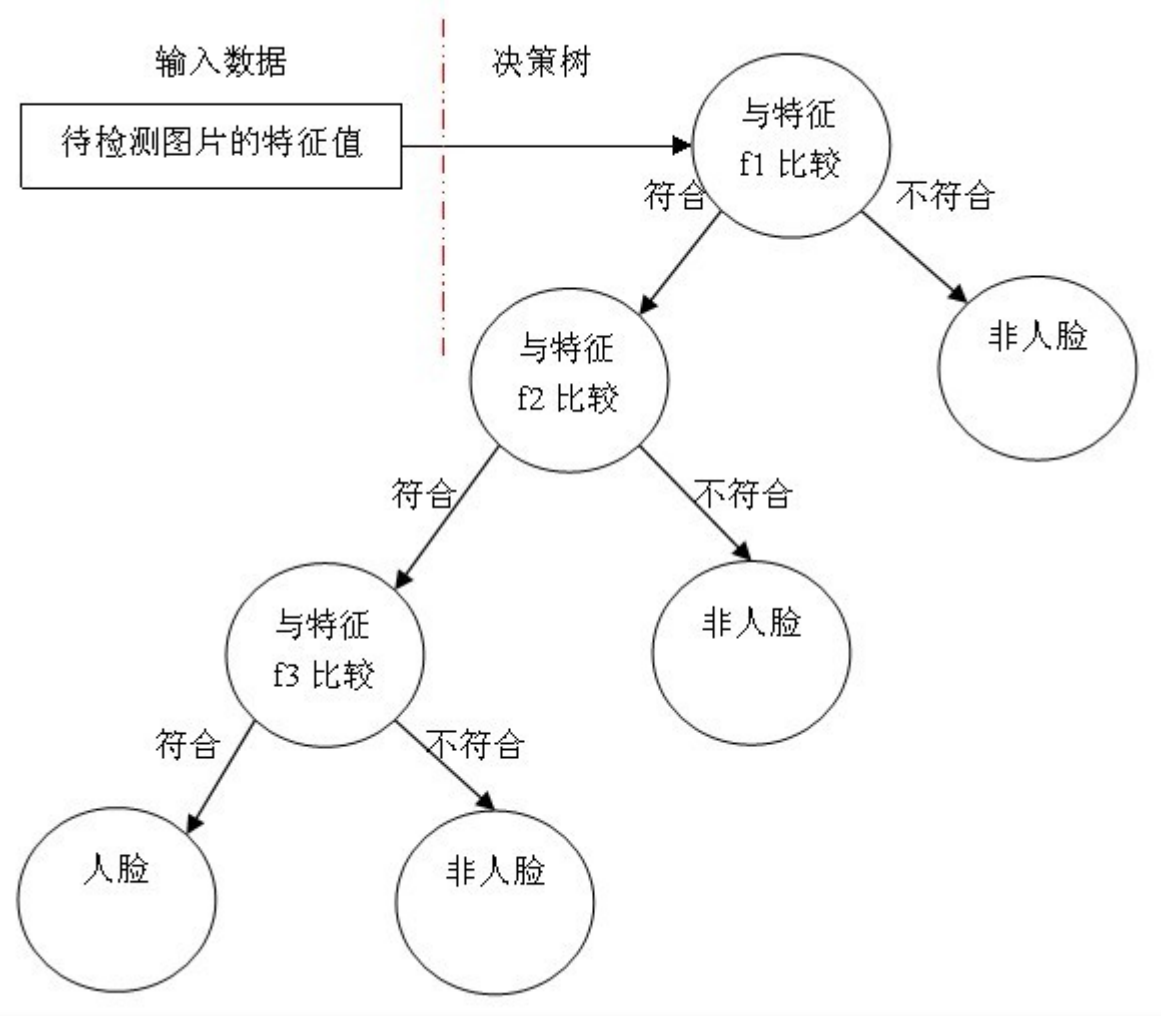
人脸定位是人脸检测、识别等一系列后续功能的基础。比如，美颜相机的贴纸、大眼等功能都需要先定位到人脸才能进行后续的处理，否则你放大的就不是眼睛，可能是一些什么不可描述的部位了。另外在人脸识别方面，比如支付宝刷脸支付、张学友演唱会抓逃犯都运用到了人脸定位然后识别的功能。

级联分类器

从一堆弱分类器里面，挑出一个最符合要求的弱分类器，用着这个弱分类器把不想要的数据剔除，保留想要的数据然后再从剩下的弱分类器里，再挑出一个最符合要求的弱分类器，对上一级保留的数据，把不想要的数据剔除，保留想要的数据。最后，通过不断串联几个弱分类器，进过数据层层筛选，最后得到我们想要的数据。这就是**级联分类器。**弱分类器分类正确率较低，但是较容易获得，强分类器分类正确率较高，但是较难获得。只要样本充足，弱

学习可以通过一定的组合获得任意精度的强学习。级联分类器就是 N个弱分类器 以级联的方式，从简单到复杂逐步串联而成。

可以用决策树来构建一个简单的弱分类器， 将提取到的特征与分类器的特征进行逐个比较，从而判断该特征是否属于人脸：



而强分类器相当于先让各个弱分类器进行投票，然后让投票结果根据各弱分类器的错误率进行加权相加，最后与平均的投票结果进行比较得到最终结果。

我们将使用OpenCV中提供的**LBP特征提取**与**Adaboost算法**（机器学习算法）训练人脸定位级联分类器。

训练

http://www.opencv.org.cn/opencvdoc/2.3.2/html/doc/user_guide/ug_traincascade.html

> 正样本：包含人脸的图片,灰度图。 正样本大小：统一大小并且不能小于负样本 负样本：不包含人脸的图片
> 负样本大小：无所谓
>
> 正、负样本个数比列大约为 1： 3

制作正样本

正样本目录为：lance，内容为100张大小为24x24的人脸

假设目录结构如下：

```
/lance
  0.jpg
  1.jpg
```

制作lance.data文件，文件内容如下：

```
#分别表示 1张人脸 ； 人脸从 0,0坐标开始；大小为24x24
lance/0.jpg 1 0 0 24 24

#==================================================================
#假设为2个人脸；  则数据为人脸分别为 100,200处的50x50 和 50,30处的25x25为人脸
lance/1.jpg 2 100 200 50 50   50 30 25 25
```

> lance.data文件内容可以使用java 程序来制作

```
public class GeneateFile{

    public static void main(String[] args) throws Exception{
        FileOutputStream fos = new
FileOutputStream("F:/Lance/OpenCV/face/train/samples/lance.data");
        for(int i =0 ;i<100;i++){
            String content = String.format("lance/%d.jpg 1 0 0 24 24\n",i);
            fos.write(content.getBytes());
        }
        fos.close();
    }
}
```

在lance.data目录执行：

> opencv_createsamples.exe 见附录一

```
opencv_createsamples -info lance.data -vec lance.vec  -num 100  -w 24 -h 24
-info: 正样本描述
-vec : 输出的正样本文件
-num : 正样本数量
-w -h: 输出样本的大小
#输出: Done. Created 100 samples  表示成功生成100个样本
```

制作负样本

负样本大小无所谓，个数为300。假如目录结构如下：

```
/bg
  0.jpg
  1.jpg
bg.data
```

则bg.data文件中的内容将如下所示：

```
bg/0.jpg
bg/1.jpg
```

训练

创建一个data 目录，执行：

```
opencv_traincascade -data data  -vec lance.vec -bg bg.data -numPos 100 -numNeg 300  -numStages
15  -featureType LBP -w 24 -h 24

-data ：  目录，需要手动创建，生成的结果 训练的模型会输出到这个目录
-vec  ：  正样本
-bg   ：  负样本
-numPos ：每级分类器训练时所用到的正样本数目
-numNeg ：每级分类器训练时所用到的负样本数目，可以大于-bg数目
-numStages：训练分类器的级数，如果层数多，分类器的误差就更小，但是检测速度慢。(15-20)
-featureType: LBP
-w -h

输出：
Training until now has taken 0 days 0 hours 0 minutes 10 seconds.
表示成功

输出：
Required leaf false alarm rate achieved. Branch training terminated.
表示成功,但是误检率已经达标。(样本太少了，生成的模型质量不行)

输出：
Bad argument < Can not get new positive sample. The most possible reason is insufficient count
of samples in given vec-file.
则意味着错误。

#参数：(未使用)
minHitRate：分类器的每一级希望得到的最小检测率。当设置为0.995(默认)时，如果numPos个数为1000个，那么第一
级分类器其中的5个就很可能不被检测，第二级选择的时候必须多选择后面的5个，按照这种规律我们为后面的每级多增
加numPos*minHitRate，5个正样本.
实际准备的正样本数量应该（读入vec的正样本数） >= numPos + (numStage - 1) * numPos * (1 - minHitRate)
按照此公式计算， numPos+14* numPos * 0.005 = 1.07*numPos ,也就是正样本数量要大于等于 1.07*numPos。
即：numPos：100，正样本数则为 107。

而我们正样本是100，所以numPos应该传：100/1.07=93。
```

因为实际设置numPos时可以将其设置的稍微再大些，最终的目的是要尽量让所有的正样本都参与到训练中。但是，过大就会出错。

由于此处我只拿了自己的脸来训练，所以模型只能定位Lance的脸.....

采集样本

```
        for (Rect face : faces) {
            //画矩形
            //分别指定 bgra
            rectangle(img, face, Scalar(255, 0, 255));
#if 0
            //使用opencv自带的模型 记录你的脸作为样本
            //把找到的人脸扣出来
            Mat m;
            //把img中的脸部位拷贝到m中
            img(face).copyTo(m);
            //把人脸 重新设置为 24x24大小的图片
            resize(m, m, Size(24, 24));
            //转成灰度
            cvtColor(m, m, COLOR_BGR2GRAY);
            char p[100];
            sprintf(p, "D:/Lance/ndk/lsn27_opencv_face/资料/img/info/%d.jpg",i++);
            //把mat写出为jpg文件
            imwrite(p,m);
#endif
        }
```

附录一：训练程序

在制作正样本时，我们会使用到这些程序。然而在我们下载的4.X版本中并没有。

直接下载OpenCV3.4.9 : https://github.com/opencv/opencv/releases

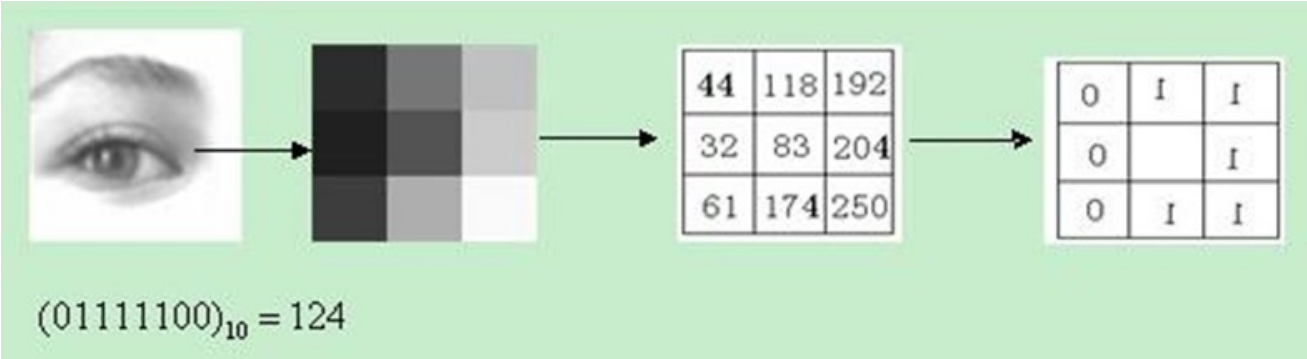
然后配置环境变量PATH增加：`opencv3\opencv\build\x64\vc15\bin`。

附录二：LBP

> LBP（Local Binary Pattern，局部二值模式）是一种用来描述图像局部纹理特征的算子,具有多分辨率、灰度尺度不变、旋转不变等特性。主要用于特征提取中的纹理提取。

> 使用LBP作为人脸检测的特征提取方式具有：计算量小；存储空间小；计算过程简单，没有复杂的除法和特殊运算，便于硬件实现；检测的时间短，检测的实时性好。

LBP的核心思想就是：以中心像素的灰度值作为阈值，与他的领域相比较得到相对应的二进制码来表示局部纹理特征。



基本LBP

原始的LBP算子定义为在3*3的窗口内，处理83这个像素点的lbp值：

将83与包围83的8个位置进行比较。如果大于83则取值为1，否则为0,然后将这些数据顺时针组合在一起，这样的到一个01111100,即：0x7C(124)。这就是83位置的lbp值。顺序并无硬性要求，只要在同一处理中保持相同的顺序即可。提取的LBP算子在每个像素点都可以得到一个LBP值，对一幅图像提取其原始的LBP算子之后，得到的原始LBP特征依然是“一幅图片”（记录的是每个像素点的LBP值）这种图片称之为lbp图谱。

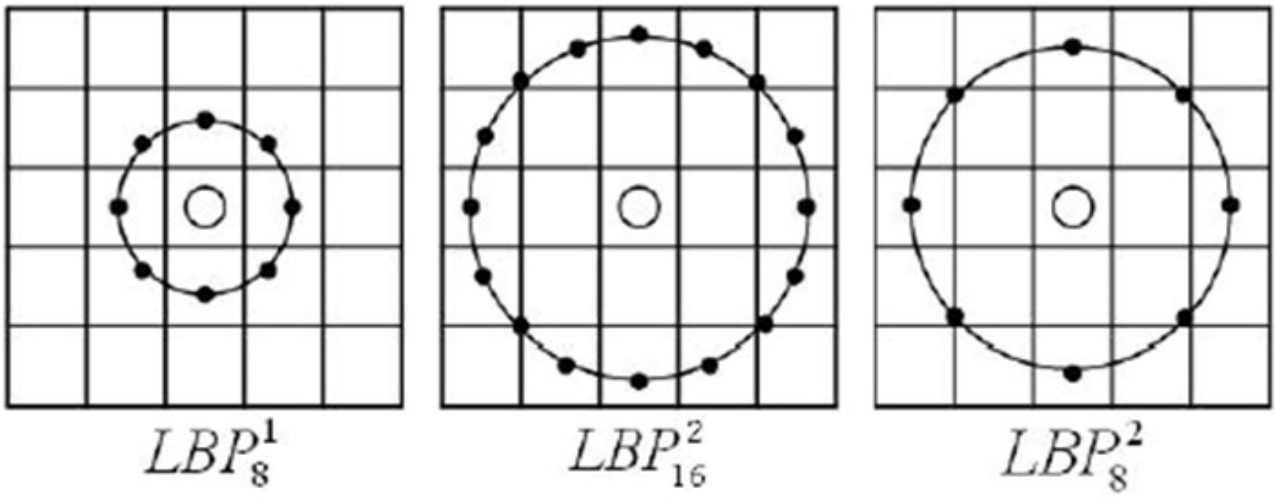
```
/**
 * 原始lbp：应该是3x3的src（先不管3x3）
 * src：原图
 * dst：计算出的lbp图谱
 *
*/
void processLBP(Mat src, Mat &dst){
    // 循环处理图像数据
    for(int i=1; i < src.rows-1;i++) {
        for(int j=1;j < src.cols-1;j++) {
            uchar lbp = 0;
            uchar center = src.at<uchar>(i,j);
            //取出对应 高、宽位置的像素 与 中心点位置进行比较
            if(src.at<uchar>(i-1,j-1)>center) { lbp += 1 << 7;}
            if(src.at<uchar>(i-1,j  )>center) { lbp += 1 << 6;}
            if(src.at<uchar>(i-1,j+1)>center) { lbp += 1 << 5;}
            if(src.at<uchar>(i  ,j+1)>center) { lbp += 1 << 4;}
            if(src.at<uchar>(i+1,j+1)>center) { lbp += 1 << 3;}
            if(src.at<uchar>(i+1,j  )>center) { lbp += 1 << 2;}
            if(src.at<uchar>(i+1,j-1)>center) { lbp += 1 << 1;}
            if(src.at<uchar>(i  ,j-1)>center) { lbp += 1 << 0;}
            dst.at<uchar>(i-1,j-1) = lbp;
        }
    }
}
//读取一张图片
```

```
Mat img = imread("/path/x.png");
cvtColor(img, img, COLOR_BGR2GRAY);
//计算lbp图谱
Mat lbp = Mat(img.rows-2, img.cols-2,CV_8UC1);
processLBP(img,lbp);
```

从IBP定义可以看出LBP是灰度不变的，简单来说就是对图像的灰度值根据一个系数X进行修改，得出的LBP值不变。

圆形LBP

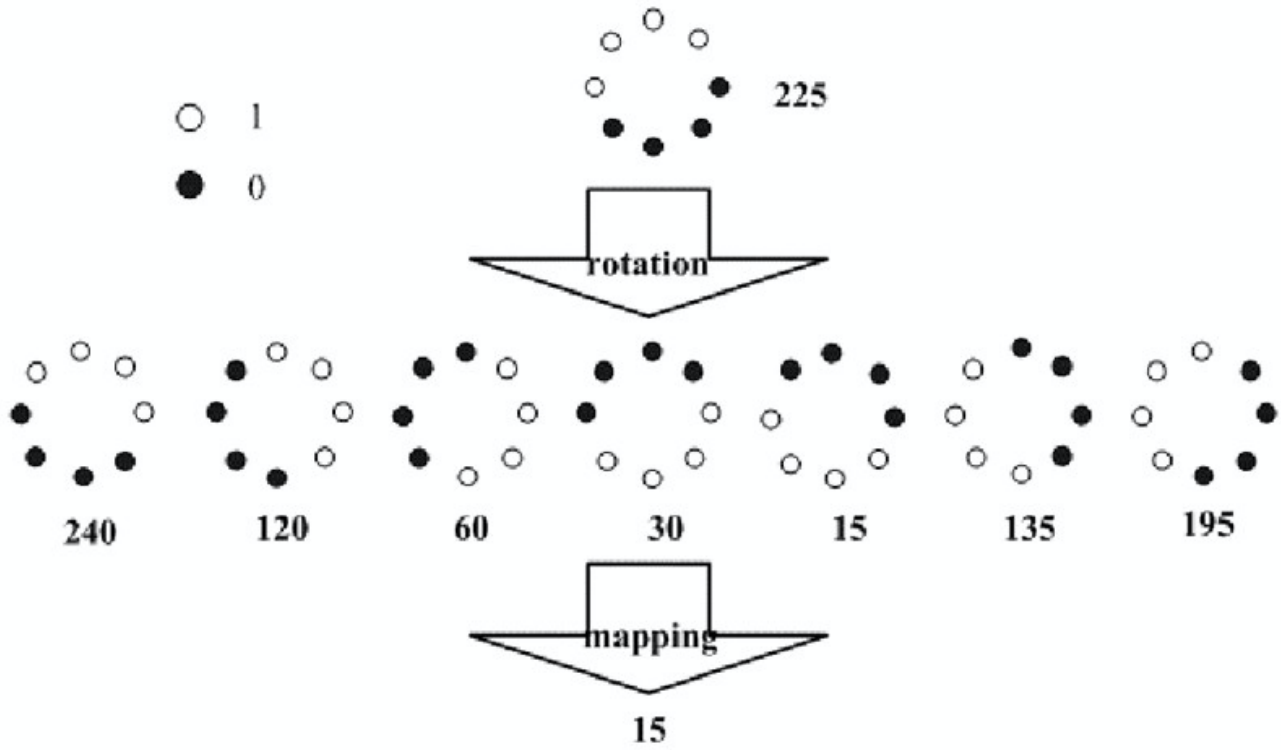
基本的 LBP算子的最大缺陷在于它只覆盖了一个固定3x3范围内的小区域，为了满足不同尺寸的需要，并达到灰度和旋转不变性的要求，对 LBP 算子进行了改进，将 3×3邻域扩展到任意邻域，并用圆形邻域代替了正方形邻域。以某个像素点为中心，以一个任意大小半径R画一个圆，将落在圆内的P个像素与中心点像素比较得到LBP算子。



旋转不变LBP

LBP 算子是灰度不变的，但却不是旋转不变的，图像的旋转就会得到不同的 LBP值。

为了解决这个问题，将 LBP算子进行了扩展具有旋转不变性。LBP的旋转不变模式，即不断旋转圆形邻域得到一系列初始定义的 LBP值，取其最小值作为该邻域的 LBP 值。



首先获得LBP值为: 11100001 (255),进过旋转分别得到8中不同的LBP值，最终得到的具有旋转不变性的 LBP值为15。

等价LBP

对于半径为R的圆形区域内含有P个采样点的LBP算子将会产生2^P(0和1的排列组合)种模式。随着邻域集内采样点数的增加，二进制模式的种类是以指数形式增加的。这么多的二进制模式导致在人脸检测时候LBP模式统计直方图过于稀疏(见下面检测原理部分)。因此需要对原始的LBP模式进行降维，也就是减少数据量。

等价模式(均匀模式)就是解决这个问题的。**在实际图像中，绝大多数LBP模式最多只包含两次从1到0或从0到1的跳变。**因此，等价模式定义为：当某个LBP所对应的循环二进制数从0到1或从1到0最多有两次跳变时，该LBP所对应的二进制就称为一个等价模式类，除等价模式类以外的模式都归为另一类，称为混合模式类。

00000000（0次跳变），00000111（1次从0到1的跳变），10001111（1到0，0到1，两次跳变）是等价模式类。

10010111（共四次跳变）是混合模式类。

通过这样的改进，二进制模式数量由原来的2^P种减少为 P* (P-1)+2种。

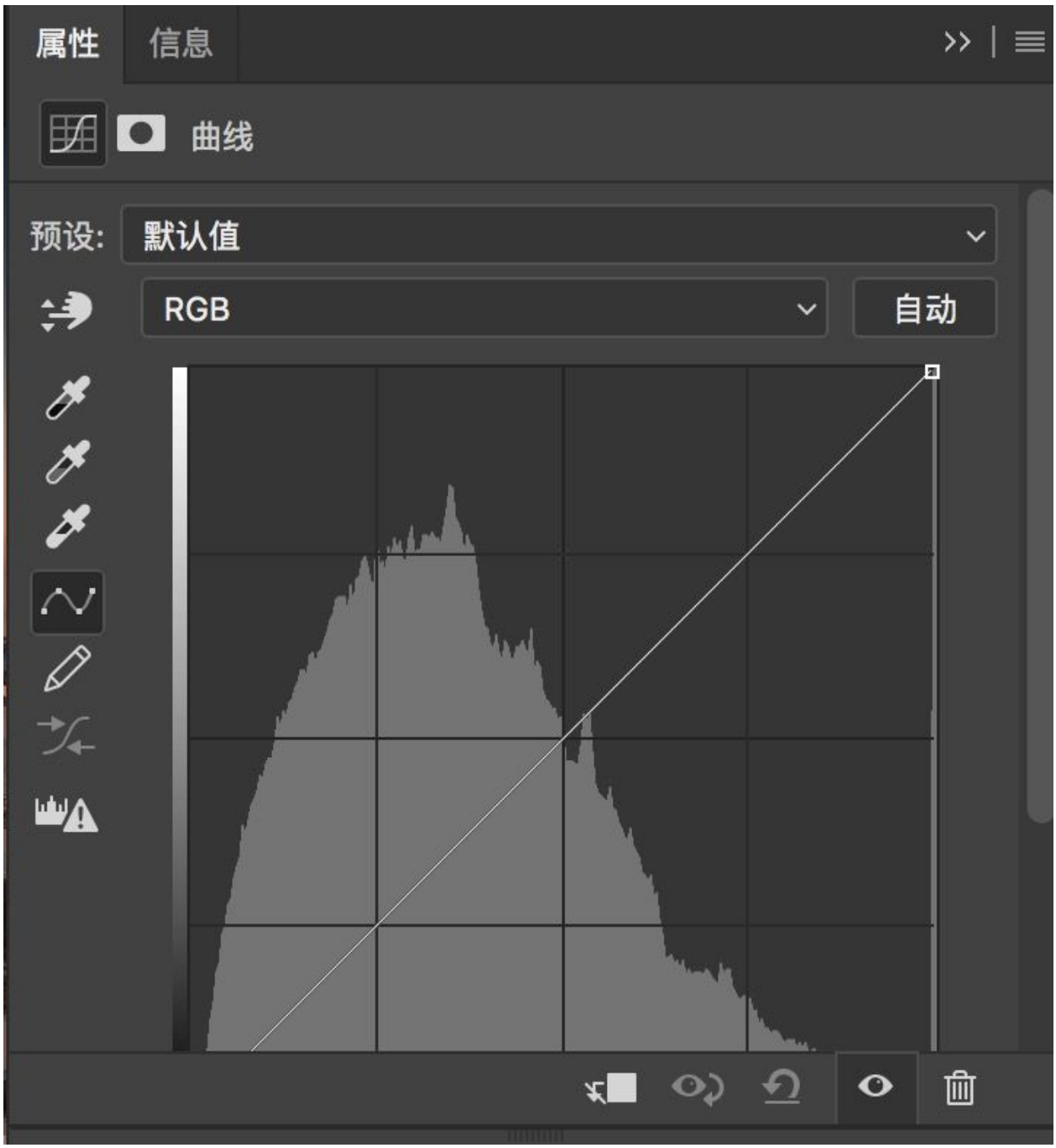
> 比如：3x3的8采样本来有256种，现在变成58(等价模式)+1种(混合模式)。即本来lbp值为0-255，也就是256种结果，转化为了59种。混合模式编码为0，等价模式根据值大小编码为1—58。
>
> 00000000 : 1
>
> 00000001 : 2
>
> 00000010 : 3
>
> 00000101 : 0 (跳变3次)

> 58种情况如下：
>
> 0次跳变：
>
> 11111111 00000000
>
> 1次跳变(14个)： 01111111 00111111 00011111 00001111 00000111 00000011 00000001
>
> 2次跳变(42个)
>
> 1：
>
> 01000000 00100000 00010000 00001000 00000100 00000010
>
> 2:
>
> 01100000 00110000 00011000 00001100 00000110
>
> 3:
>
> 01110000 00111000 00011100 00001110
>
> 4:
>
> 01111000 00111100 00011110
>
> 4:
>
> 01111100 00111110
>
> 6:
>
> 01111110

检测原理

将一幅图片划分为若干的子区域，对每个子区域内的每个像素点都提取LBP特征，然后，在每个子区域内建立LBPH（LBP特征的统计直方图）。 每个子区域就可以用一个统计直方图来进行描述；整个图片就由若干个统计直方图组成 之后，将图片和人脸的直方图进行相似性比较。

> 直方图:
>
> 把图片的亮度分为0到255共256个数值，数值越大，代表的亮度越高。其中0代表纯黑色的最暗区域，255表示最亮的纯白色，而中间的数字就是不同亮度的灰色。用横轴代表0-255的亮度数值。竖轴代表照片中对应亮度的像素数量，这个函数图像就被称为直方图。
>
> 简单来说，图像的直方图是用来表现图像中亮度分布的情况,给出的是图像中某个亮度或者某个范围亮度下共有几个像素

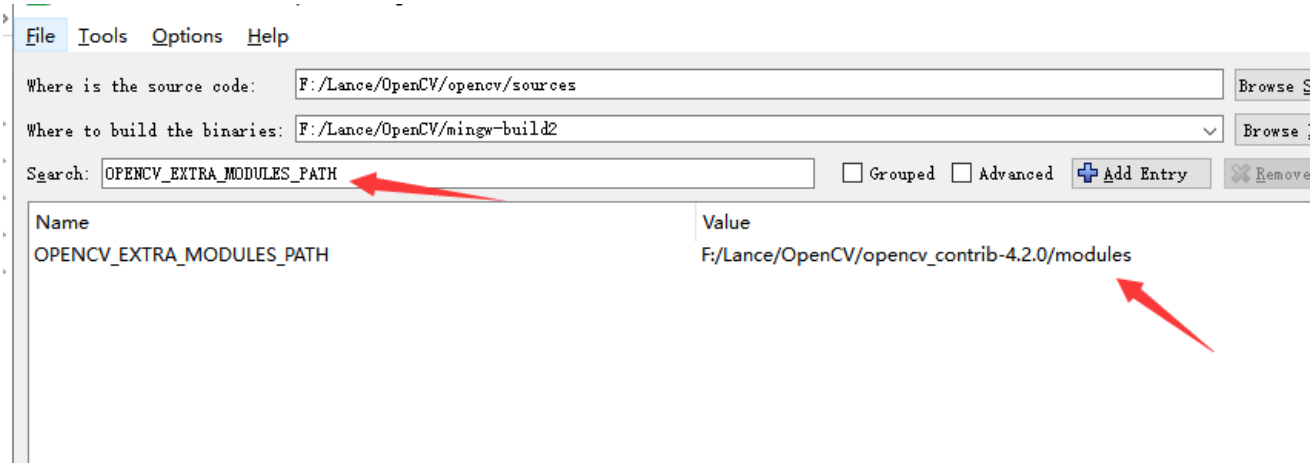


附录三：人脸识别

OpenCV提供了 `cv::face::FaceRecognizer` 帮助我们进行人脸识别，但是它位于 `opencv_contrib` 模块中，因此如果需要使用，那么需要将Opencv与opencv_contrib共同编译。

下载：https://github.com/opencv/opencv_contrib

在编译OpenCV时，搜索：`OPENCV_EXTRA_MODULES_PATH` 并配置 contrib的解压目录



然后执行编译视频中的流程即可。 在配置阶段可能因为某些依赖文件下载失败。根据提示手动下载或者挂上代理重试即可。

编译完成之后，即可使用：

```
#include <opencv2/face.hpp>
int main() {

#if 1
    Ptr<cv::face::FaceRecognizer> model = cv::face::EigenFaceRecognizer::create();
```

可参考：https://www.cnblogs.com/llguanli/p/7286615.html