人脸识别

曹鹏霄1，李原丞2

（1.宁波工程学院，浙江 宁波 315211；2.宁波工程学院，浙江 宁波 315211）

摘要**：**人脸识别是当下比较火热的一门技术，为了解其中的相关原理和算法，掌握其中的窍门，在了解其中原理的基础上，用python及MATLAB编写了相关不同种类算法的代码。整个项目主要包含了两大方面内容——“人脸检测”和“人脸识别”。我们先识别出相关人脸，其次训练样本数据，最后再用测试集对其进行检测以达到相关的精度要求。最后能保持在一定的准确度的基础上识别人脸。本算法主要是基于Opencv库和深度学习算法。

关键字**：**人脸检测；特征提取；人脸识别；图像预处理；深度学习；

**中图分类号：**

**Research on face recognition**

CAO PengXiao1,LI YuanCheng2

(1.Ningbo university of technology,Ningbo 315211;2.Ningbo university of technology,Ningbo 315211;)

**Abstract:** Face recognition is a hot technology at present. In order to understand the relevant principles and algorithms and master the tips, based on the understanding of the principles, we use Python and MATLAB to write the codes of different kinds of algorithms. The whole project mainly includes two aspects - "face detection" and "face recognition". We first recognize the relevant face, then train the sample data, and then use the test set to detect it to achieve the relevant accuracy requirements. Finally, face recognition can be maintained on the basis of certain accuracy. This algorithm is mainly based on OpenCV library and deep learning algorithm.

**Key words:**Face detection: feature extraction: face recognition: image preprocessing:

# 简介

MADE2项目是为了解人脸识别的相关算法和其中的数学原理，提高自身的数学分析和代码的分析能力，使用python和MATLAB编写了相关的识别程序，再通过程序对人脸进行采集，分析，训练和识别。从中我们也了解了不同算法的采集，分析，训练和识别的数学原理，并加以自己的理解将其完成了，从中我们了解学习了很多关于人脸识别的知识和原理，对我们的数学思维尤其是对于线性代数方面的知识和应用有了更深的理解，我们小组主要采用两中算法实现人脸识别——1是PCA算法，2是LBP算法。

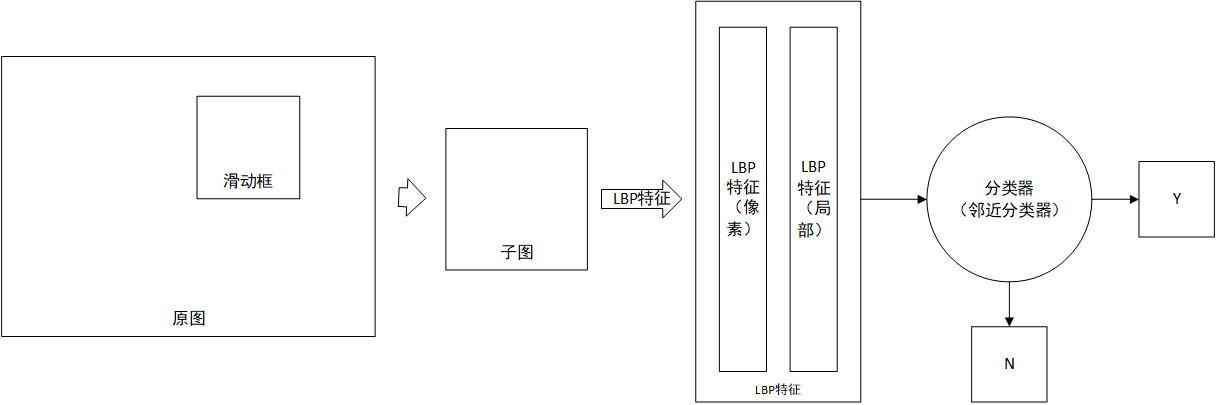
# LBP算法及其推导过程

## LBP背景以及理论基础

人脸识别是指将一个需要识别的人脸和人脸库中的某个人脸对应起来（类似于指纹识别），目的是完成识别功能，该术语需要和人脸检测进行区分，人脸检测是在一张图片中把人脸定位出来，完成的是搜寻的功能。从OpenCV2.4开始，加入了新的类FaceRecognizer，该类用于人脸识别，使用它可以方便地进行相关识别实验。

原始的LBP算子定义为在3\*3的窗口内，以窗口中心像素为阈值，将相邻的8个像素的灰度值与其进行比较，若周围像素值大于或等于中心像素值，则该像素点的位置被标记为1，否则为0。这样，3\*3邻域内的8个点经比较可产生8位二进制数（通常转换为十进制数即LBP码，共256种），即得到该窗口中心像素点的LBP值，并用这个值来反映该区域的纹理特征。

LBP系统结构如下图所示：

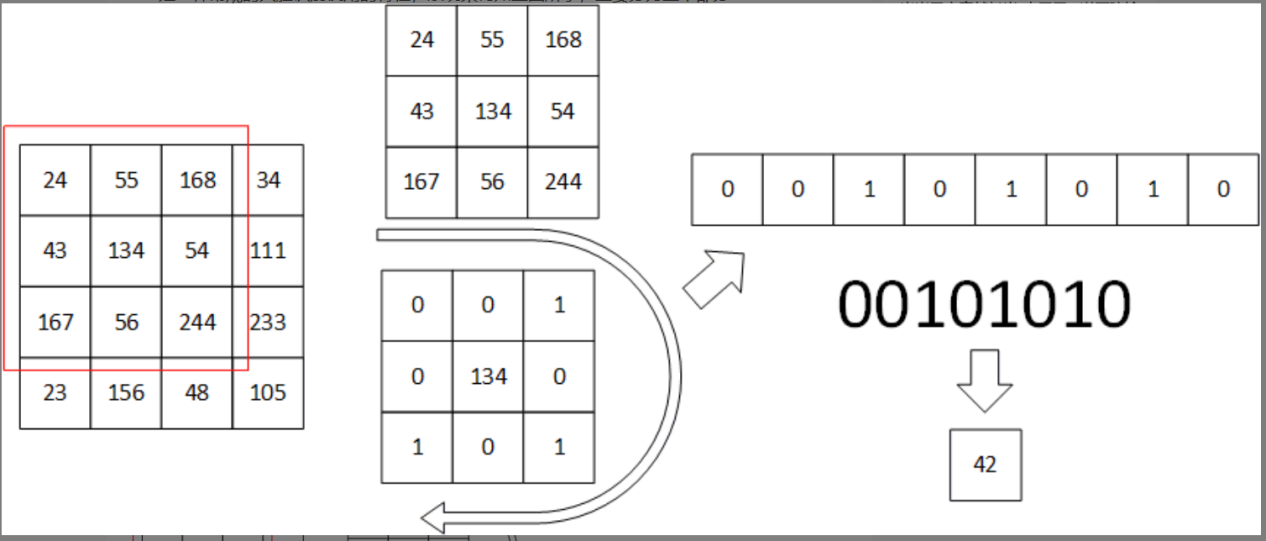


(1).滑动框：滑动框在图片上滑动，产生不同的子图

(2).LBP特征提取器：针对滑动框产生的子图，计算LBP特征

(3).分类器：根据LBP特征，判断当前图片是否是人脸

原始LBP特征是一个3X3区域的区域特征。考虑一个像素的特征值，该特征值与周围的8个像素（3X3区域）有关，对于像素值大于该像素的周围像素赋值1，其他赋值0，如下图所示：

。

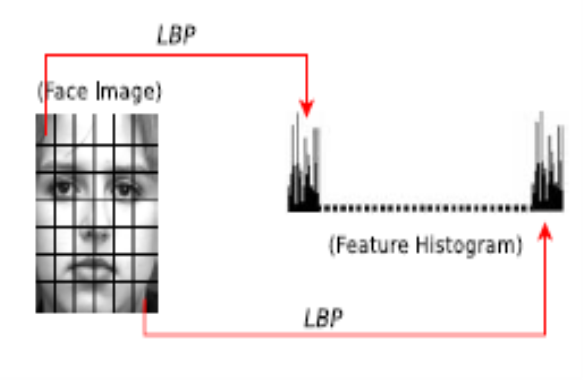
用公式表示就是：



其中是中心像素，是灰度值，是相邻像素的灰度值，是一个符号函数。

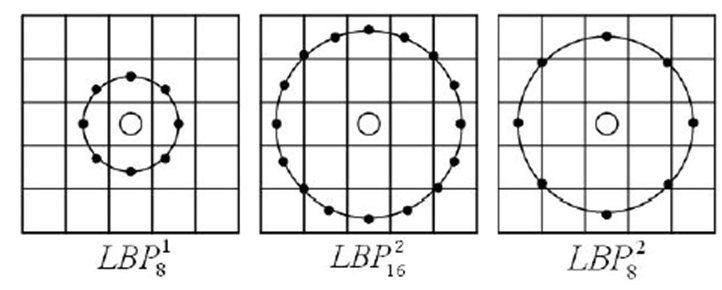


得到了每个像素分割的LBP。建立一个直方图，每个LBP的种类作横轴，每种LBP出现的次数为纵轴。下图就可以反映该区域的纹理信息。

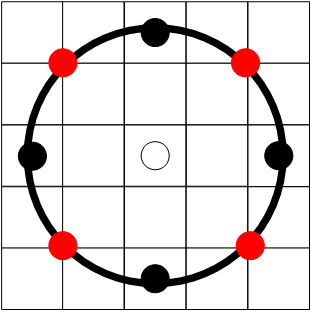


## 算法优化采用圆形LBP算子

基本的 LBP算子的最大缺陷在于它只覆盖了一个固定半径范围内的小区域，这显然不能满足不同尺寸和频率纹理的需要。为了适应不同尺度的纹理特征，并达到灰度级和旋转不变性的要求，Ojala等对 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，改进后的 LBP算子允许在半径为 R的圆形邻域内有任意多个像素点。从而得到了诸如半径为R的圆形区域内含有P个采样点的LBP算子。下图为几种圆形LBP算子，上标表示半径，下标表示所选取的算子数量。



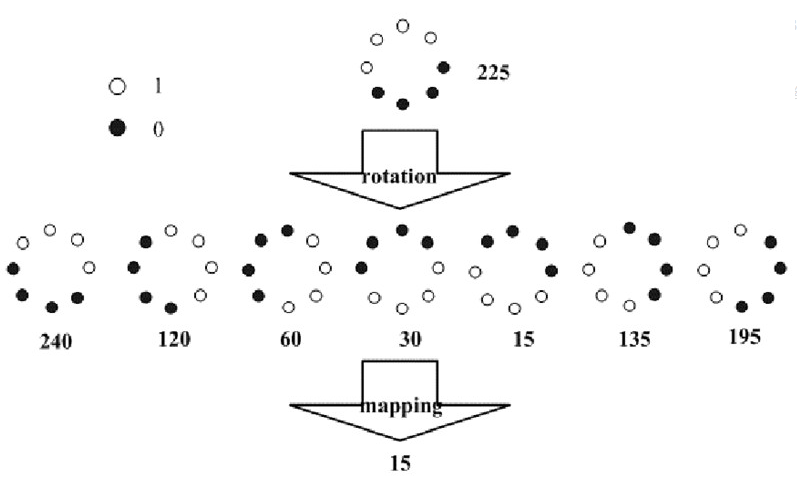
圆形LBP特征与原始LBP特征类似，只是周围像素为一个圆形环绕的，如下图所示：



LBP使用如图所示8个实心点位置的像素计算，其中四个红色实心点不落在像素上，使用双线性插值的方法计算对应的值，最后带入原始LBP的计算方法中计算中间像素的值。该LBP使用(P,R)表示，P表示带入计算的像素点数，R表示半径，如上图即为.

## LBP的旋转不变模式

从 LBP 的定义可以看出，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。   
　　Maenpaa等人又将 LBP 算子进行了扩展，提出了具有旋转不变性的 LBP 算子，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。   
　　下图给出了求取旋转不变的 LBP 的过程示意图，图中算子下方的数字表示该算子对应的 LBP 值，图中所示的 8 种 LBP模式，经过旋转不变的处理，最终得到的具有旋转不变性的 LBP 值为 15。也就是说，图中的 8 种 LBP 模式对应的旋转不变的 LBP 模式都是00001111。



即定义：



其中，表示将二进制特征向右循环了次。

通过引入旋转不变的定义，LBP 算子不仅对于图像旋转表现得更为鲁棒，并且 LBP 模式的种类进一步减少，使得纹理识别更加容易。从另一方面来说，旋转不变的 LBP 算子的最大缺陷在于其丢失了方向信息，而对于某些场合来说，方向信息是十分重要的信息。然而，对于同类纹理图像的分析，算子已经被证明是有效的。

## LBP等价模式

一个LBP算子可以产生不同的二进制模式，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P2种模式。很显然，随着邻域集内采样点数的增加，二进制模式的种类是急剧增加的。例如：5×5邻域内20个采样点，有220＝1,048,576种二进制模式。如此多的二值模式无论对于纹理的提取还是对于纹理的识别、分类及信息的存取都是不利的。同时，过多的模式种类对于纹理的表达是不利的。例如，将LBP算子用于纹理分类或人脸识别时，常采用LBP模式的统计直方图来表达图像的信息，而较多的模式种类将使得数据量过大，且直方图过于稀疏。因此，需要对原始的LBP模式进行降维，使得数据量减少的情况下能最好的代表图像的信息。

为了解决二进制模式过多的问题，提高统计性，Ojala提出了采用一种“等价模式”（Uniform Pattern）来对LBP算子的模式种类进行降维。Ojala等认为，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，Ojala将“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。如00000000（0次跳变），00000111（只含一次从0到1的跳变），10001111（先由1跳到0，再由0跳到1，共两次跳变）都是等价模式类。除等价模式类以外的模式都归为另一类，称为混合模式类，例如10010111（共四次跳变）

通过这样的改进，二进制模式的种类大大减少，而不会丢失任何信息。模式数量由原来的2P种减少为 P ( P-1)+2种，其中P表示邻域集内的采样点数。对于3×3邻域内8个采样点来说，二进制模式由原始的256种减少为58种，这使得特征向量的维数更少，并且可以减少高频噪声带来的影响。

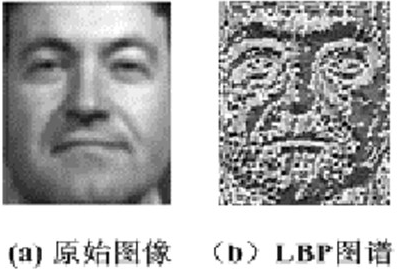
正式定义如下：



其中，即顺时针方向每个局部领域像素的的二进制的变化次数。

## LBP特征用于检测的原理

显而易见的是，上述提取的LBP算子在每个像素点都可以得到一个LBP“编码”，那么，对一幅图像（记录的是每个像素点的灰度值）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）**。**



对图像采用LBP算子提取特征后得到的仍然是一幅图像，经过LBP运算后，所改变的只是各个像素点的取值，我们将经过LBP运算后的图像称为LBP图谱。在LBP的应用中，如纹理分类、人脸分析等，一般都不将LBP图谱作为特征向量用于分类识别，而是采用LBP特征谱的统计直方图作为特征向量用于分类识别。常用非参数统计的方法比较两个样本间LBP直方图分布的相似性，采用这种方法的优点在于可以避免对特征分布做假设。

  因为，从上面的分析我们可以看出，这个“特征”跟位置信息是紧密相关的。直接对两幅图片提取这种“特征”，并进行判别分析的话，会因为“位置没有对准”而产生很大的误差。后来，研究人员发现，可以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LBP特征的统计直方图。如此一来，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；

        例如：一幅100\*100像素大小的图片，划分为10\*10=100个子区域（可以通过多种方式来划分区域），每个子区域的大小为10\*10像素；在每个子区域内的每个像素点，提取其LBP特征，然后，建立统计直方图；这样，这幅图片就有10\*10个子区域，也就有了10\*10个统计直方图，利用这10\*10个统计直方图，就可以描述这幅图片了。之后，我们利用各种相似性度量函数，就可以判断两幅图像之间的相似性了。

1.6对LBP特征向量进行提取的步骤

(1).首先将检测窗口划分为16×16的小区域（cell）；

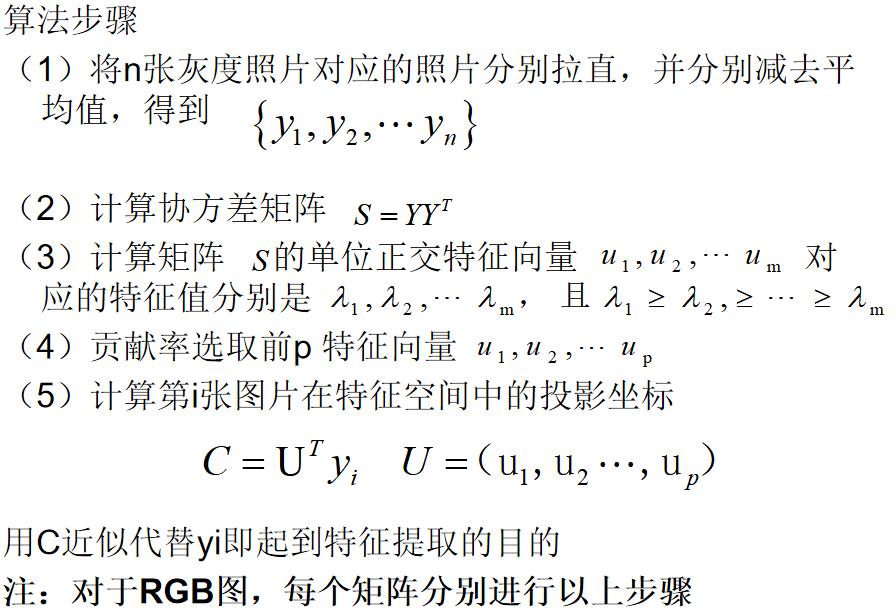
(2)对于每个cell中的一个像素，将相邻的8个像素的灰度值与其进行比较，若周围像素值大于中心像素值，则该像素点的位置被标记为1，否则为0。这样，3\*3邻域内的8个点经比较可产生8位二进制数，即得到该窗口中心像素点的LBP值；

(3)然后计算每个cell的直方图，即每个数字（假定是十进制数LBP值）出现的频率；然后对该直方图进行归一化处理。

(3)最后将得到的每个cell的统计直方图进行连接成为一个特征向量，也就是整幅图的LBP纹理特征向量；

然后便可利用SVM或者其他机器学习算法进行分类了**。**

# PCA(主成分分析)



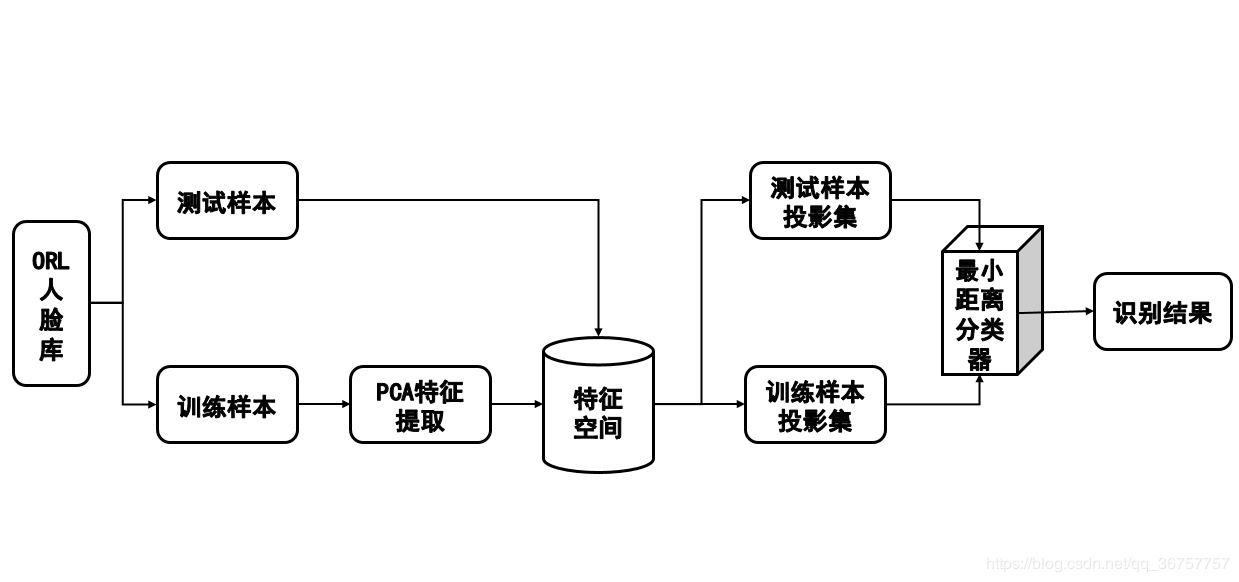
最近重构性：样本点到这个超平面的距离都足够近

最大可分性：样本点在这个超平面上的投影能尽可能分开

总的来说，我们的目标就是找到另一组基，使得同样的这些样本在那组基上可以达到方差最大，协方差最小。

优化目标：  
将一组N维向量降为K维(K大于0，小于N)，其目标是选择K个单位(模为1)正交基，使得原始数据变换到这组基上后，各字段两两间协方差为0(各自独立)，而字段的方差则尽可能大(投影后的点尽可能离散)。在正交的约束下，取最大的K个方差

2.1 PCA人脸识别的基本流程



(Figure 2-1)

从上图中可以看出PCA人脸识别分为以下几个阶段：特征提取、构造特征空间、投影计算、分类器决策，我们的程序就是根据流程图所示编写。

我们本次采取学院内30个人各20张照片作为样本，其中前十张作为训练样本，后十张作为测试样本。使用matlab进行代码编写。

2.2读入ROL库数据

通过matlab自带的读取照片的函数imread()将前十个照片转换成矩阵集所保存。并通过转置等方式得到表示各个训练图片的行向量。

2.3 PCA特征提取

2.3.1按之前所排序完成后的行向量来求出总的矩阵的平均脸

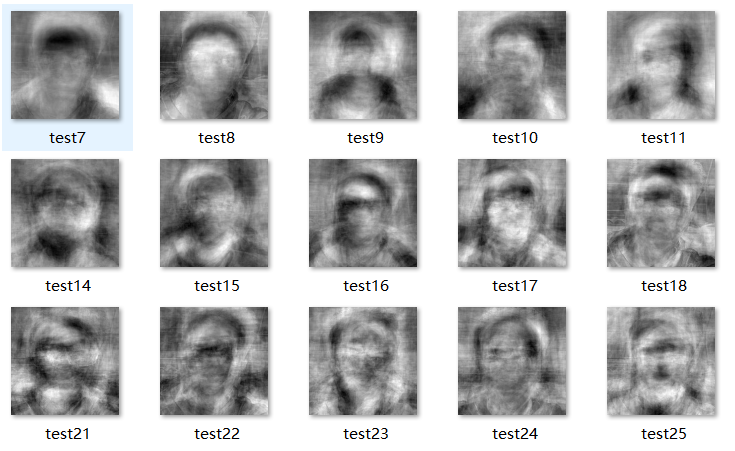


(Figure 2-2)

1. 3.2计算协方差矩阵并得到特征向量。

根据协方差矩阵公式： 

特征脸如下图（Figure 3）所示:

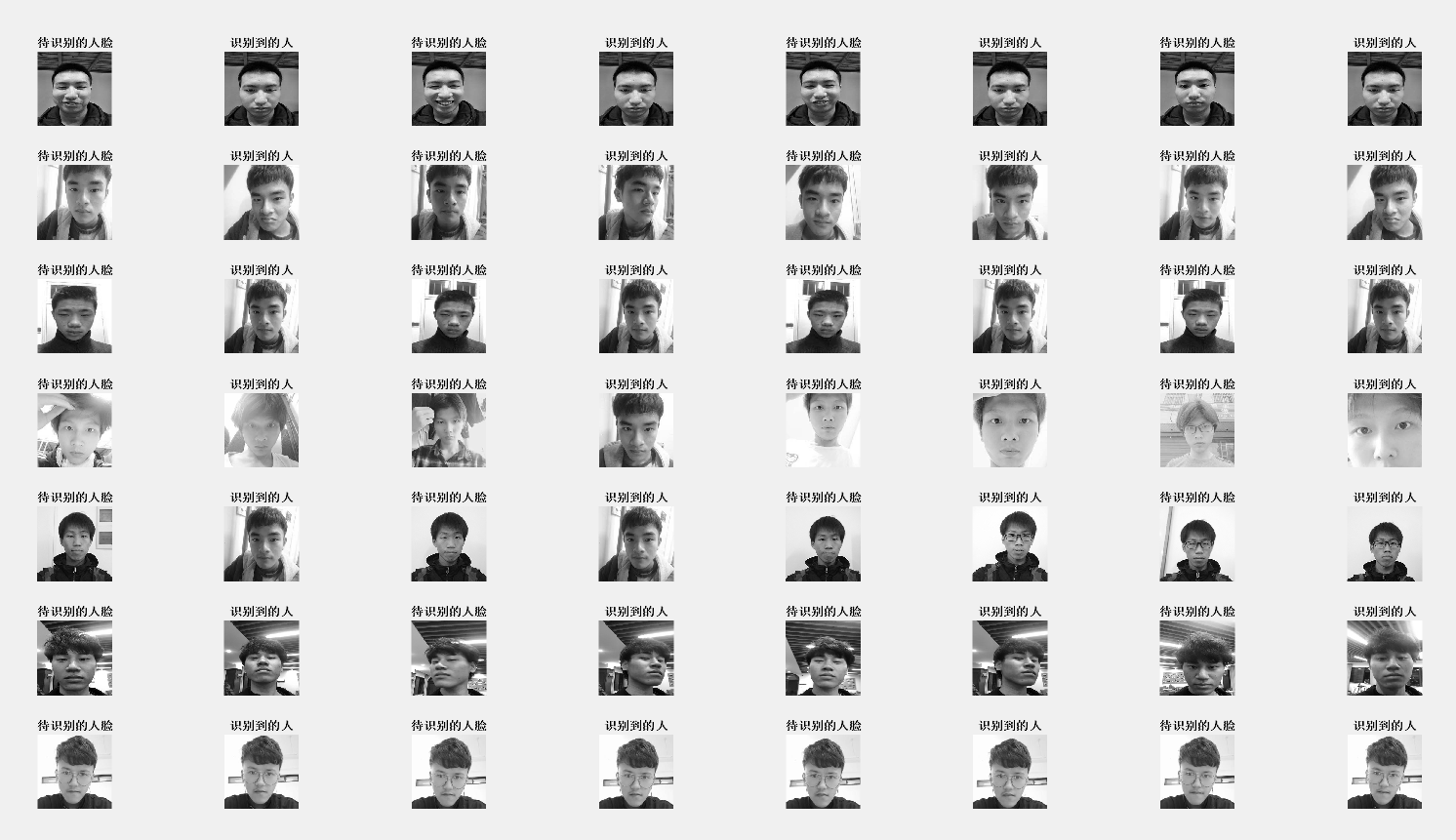


(Figure 2-3)

2.3.3对特征向量根据特征值进行从大到小排序在进行对角化处理，之后选取累计贡献率达到90%以上的特征向量，再乘上我们的初始原图像矩阵就可以得到我们所需要的正交基。

2.3.4最小距离分类器

我们最后将测试样本集做上述相同的处理与遍历的训练样本集做比对，并得到最小的差异的结果进行输出并判断识别是否准确。

运行结果为：

(Figure 2-4)

本次PCA人脸识别实验结果准确率约为75%。

* 1. 误差分析

2.4.1人脸大小不一和经过美颜的照片识别率不高，主要原因可能是照片通过滤镜等方式处理过后矩阵信息某些信息被放大导致求特征向量时提取到的干扰信息被放大了，以及样本数量过少，或者拍摄角度等问题使得训练照片与测试照片本身存在较大的差异。

2.4.2训练样本过少导致，导致训练结果不精确。

2.4.3训练样本贡献率设置不准确，使得所得正交基不够标准。

# 总论

在老师和同学还有小组成员的共同努力下，我们小组最终完成了人脸识别项目，并且顺利的结束了答辩，MADE人脸识别项目告一段落。在项目的这段时间，老师给我们安排了关于人脸识别的项目课，老师课上给我们讲解了许多的人脸识别算法的基本原理和它们的数学原理，这就更加有助于我们吃透算法，从而更好地完成代码的编写，老师所希望的就是我们了解数学原理以后，从最底层开始编写代码，我们小组也基本上做到了这一点。

老师给我们讲的人脸识别算法不止有一种，不同的算法有着不同的特点，我们小组选择的就是PCA和LBP。

PCA的特点就是降低数据集的维数，同时保持数据集中的对方差贡献最大的特征，它的优点有:(1)使得数据更容易被利用.

(2).降低算法的计算开销

(3).完全无参数限制，结果更容易被理解

缺点有：(1).如果用户对观测对象有一定的先验知识，掌握了数据的一些特征，却无法通过参数化等方法对处理过程进行干预，可能会得不到预期的效果，效率也不高.

(2).特征值分解有一些局限性，比如变换的矩阵必须是方阵

(3).在非高斯分布情况下，PCA方法得出的主元可能并不是最优的

LBP的特点就是LBP特征直方图可以作为一个图片的特征判断，它的优点有：

(1).对光照不敏感，光对图片判断的影响较小

(2).旋转不变性

(3)灰度不变性

缺点有：

(1).姿态，表情对图片识别影响比较大

(2).支持多尺度分析

(3).样本需求大

两种算法各有各的特点，它们应用十分广泛，如：高维数据的降维压缩，高维数据的探索和可视化，图像恢复，纹理分析等方面；如果学习利用好了，这将会对我们有很大的帮助，在这次的项目中我们遇到了很多困难，但我们没有退缩，在克服了一个又一个的困难之后，我们顺利完成了任务，也达到了老师的要求；我希望在以后的学习生活中，还能保持这种坚韧，执着的求知精神，也希望在以后的学习中取得更大的进步。

# 参考文献

# [1]黄非非，基于 LBP 的人脸识别研究，重庆大学硕士学位论文，2009.5

# 附录

LBP算法(python)

**import cv2**

**recognizer = cv2.face.LBPHFaceRecognizer\_create()**

**recognizer.read('data/trainner.yml')**

**faceCascade = cv2.CascadeClassifier("D:\\opencv\\opencv\\sources\\data\\haarcascades\\haarcascade\_frontalface\_default.xml")**

**font = cv2.FONT\_HERSHEY\_SIMPLEX**

**idnum = 0**

**names = ['lyc']**

**cam = cv2.VideoCapture(0)**

**minW = 0.1\*cam.get(3)**

**minH = 0.1\*cam.get(4)**

**while True:**

**ret, img = cam.read()**

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

**faces = faceCascade.detectMultiScale(**

**gray,**

**scaleFactor=1.2,**

**minNeighbors=5,**

**minSize=(int(minW), int(minH))**

**)**

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

**cv2.rectangle(img, (x, y), (x+w, y+h), (0, 255, 0), 4)**

**idnum, confidence = recognizer.predict(gray[y:y+h, x:x+w])**

**print(100-confidence)**

**if 100-confidence >40:**

**idnum = names[idnum]**

**confidence = "{0}%".format(round(100-confidence))**

**else:**

**idnum = "unknown"**

**confidence = "{0}%".format(round(100-confidence))**

**cv2.putText(img, str(idnum), (x+5, y-5), font, 1, (0, 0, 255), 4)**

**cv2.putText(img, str(confidence), (x+5, y+h-5), font, 1, (255, 255, 0), 3)**

**cv2.imshow('camera', img)**

**k = cv2.waitKey(10)**

**if k == 27:**

**break**

**cam.release()**

**cv2.destroyAllWindows()**