

浙江大学实验报告

专业：求是科学班（计算机）

姓名：蒋仕彪

学号：3170102587

日期：2019/12/10

课程名称：计算机视觉 指导老师：宋明黎 成绩：

实验名称：HW#3: Eigenface 人脸识别算法

一、实验目的和要求

自己写代码实现 Eigenface 人脸识别的训练与识别过程：

1. 假设每张人脸图像只有一张人脸，且两只眼睛位置已知（即可人工标注给出）。每张图像的眼睛位置存在相应目录下的一个与图像文件名相同但后缀名为 txt 的文本文件里，文本文件中用一行、以空格分隔的 4 个数字表示，分别对应于两只眼睛中心在图像中的位置；

2. 实现两个程序过程（两个执行文件），分别对应训练与识别

3. 自己构建一个人脸库（至少 40 人，包括自己），课程主页提供一个人脸库可选用。

4. 不能直接调用 OpenCV 里面与 Eigenface 相关的一些函数，特征值与特征向量求解函数可以调用；只能用 C/C++，不能用其他编程语言；GUI 只能用 OpenCV 自带的 HighGUI，不能用 QT 或其他；平台可以用 Win/Linux/MacOS，建议 Win 优先；

5. 训练程序格式大致为：“mytrain.exe 能量百分比 model 文件名 其他参数...”，用能量百分比决定取多少个特征脸，将训练结果输出保存到 model 文件中。同时将前 10 个特征脸拼成一张图像，然后显示出来。

6. 识别程序格式大致为：“mytest.exe 人脸图像文件名 model 文件名 其他参数...”，将 model 文件装载进来后，对输入的人脸图像进行识别，并将识别结果叠加在输入的人脸图像上显示出来，同时显示人脸库中跟该人脸图像最相似的图像。

二、实验内容和原理

2.1 构建人脸数据集

为了简化运算，我采用了黑白的人脸图片。

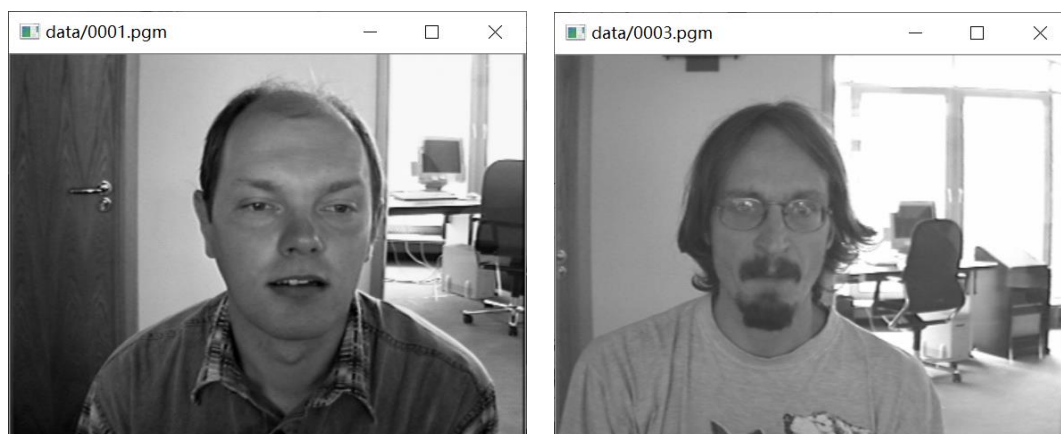
老师提供的人脸数据姿势比较单一，而且都是女性图像，于是我考虑更换一个大一点全一点的数据集。最后我采用的是 **BioID Face Database**¹ 的人脸数据集，一共有 1521 张 386*286 的灰度图像，23 个人；每张图片提供 pgm 文件和对应的 eye 坐标。

因为总共只有 23 个人，同人的数据会特别多。我就对图片做了一些预处理，用 python 将 1521 张图片随机打乱，最终做实验时，默认抽取了前 100 张。这样期望每个人只会有 5 张照片左右。代码见下。

```
id = [k for k in range(1521)]
random.shuffle(id)
for i in range(len(id)):
    os.rename("BioID_{:04d}.pgm".format(i), "{:04d}.pgm".format(id[i]))
    os.rename("BioID_{:04d}.eye".format(i), "{:04d}.eye".format(id[i]))
```

¹ <https://ftp.uni-erlangen.de/pub/facedb/readme.html>

这些人都是怎样的呢？我抽取了 2 张展示在下面：



2.2 根据人眼睛位置截取人脸框

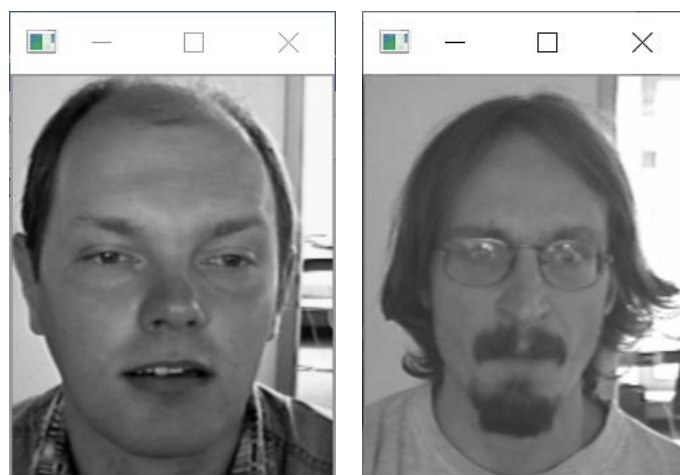
BioID Face Database 的人脸数据集有一个好处：人脸都在比较固定的位置（偏屏幕中间）。这样我们可以根据眼睛的位置并调一下参数，大致截取出人脸框。以下是我最终抠图的参数：

```
int avey = (y1 + y2) / 2;
int startx = x1 - (Xlen - (x2 - x1)) / 2;
startx = max(startx, 0); startx = min(startx, Xmax - Xlen);
int starty = avey - 90;
starty = max(starty, 0); starty = min(starty, Ymax - Ylen);
Mat crop = pic(Range(starty, starty + Ylen), Range(startx, startx + Xlen));
```

其中 $(x1, y1)$, $(x2, y2)$ 是人脸的参数。
 $Xmax=384$, $Ymax=286$, 表示原图的尺寸。最后我对人脸框的尺寸做了一个限定，它必须是 $(Xlen=200, Ylen=160)$ 。所以我要根据眼睛的位置计算出人脸框的左上角。

做了这样的框效果具体是怎么样的呢？右边展示了上面两幅图裁剪后的例子。

其实 **opencv** 里有一些基于预训练的人脸检测函数，不过鉴于本实验的“重新造轮子”的宗旨，人脸框的截取我是手工调参的。在我的参数设定里，高度 160 的图片中，眼睛上面部分：眼睛下面部分 = 90:70。

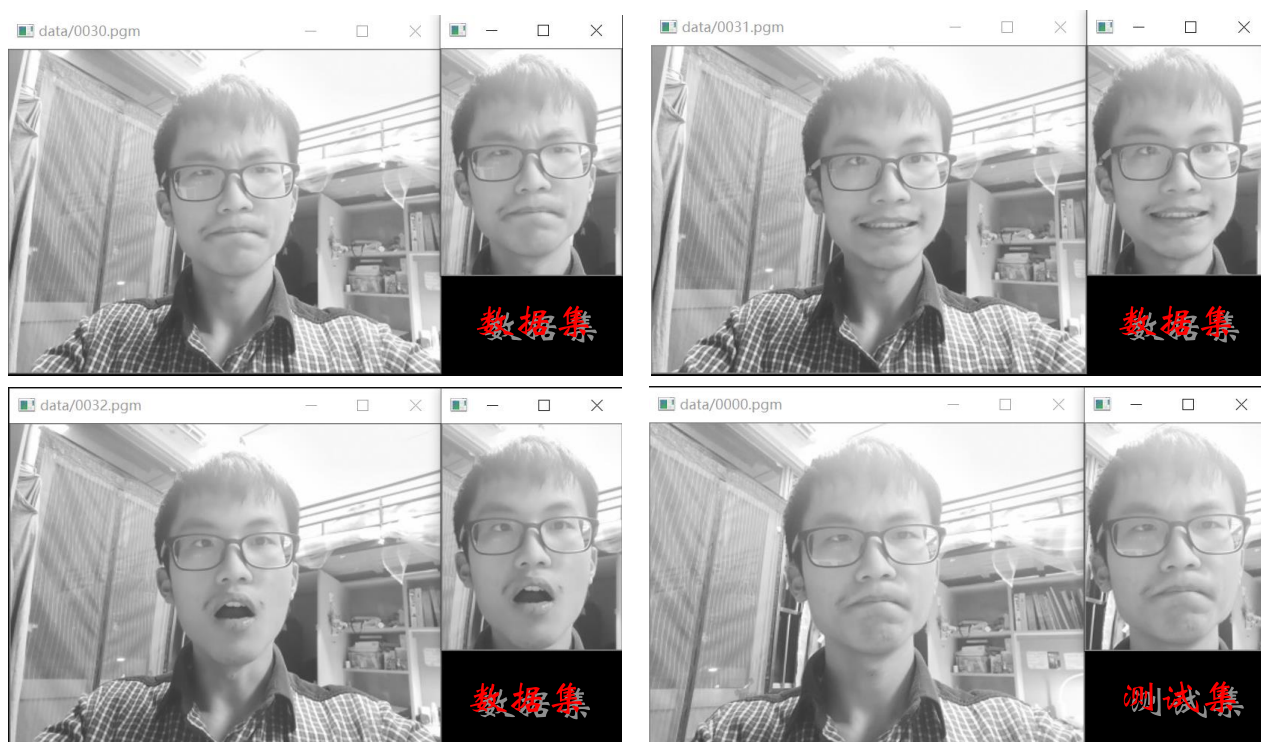


2.3 加入自己的图片

实验要求是要添加自己的图片，我就用手机自拍了 4 张，其中 3 张表情各异，最后一张和之前的某一张表情相似。拍完后，我借助 **opencv** 将其转成 **pgm** 格式，同时仿照 **BioID Face Database** 的格式，把眼睛的位置手动标了出来（用画图打开图片，鼠标悬停在眼睛上时就会显示出坐标）。

我将前 3 张图片混入 100 张的训练集里（替换原来的 0030~0032），剩下一张拿来测试。

这四张图片和裁剪后的效果如图：



2.4 PCA 降维

PCA 算法的基本原理如下：

假设 $X = \{x_1, x_2, \dots, x_n\}$ 表示一个特征向量，其中 $x_i \in \mathbb{R}^d$ 。【注： x_i 可能也是一个列向量】

1. 计算均值向量 μ

$$\mu = \frac{1}{n} \sum_{i=1}^n x_i$$

2. 计算协方差矩阵 S

$$S = \frac{1}{n} \sum_{i=1}^n (x_i - \mu)(x_i - \mu)^T$$

3. 计算 S 的特征值 λ_i 和对应的特征向量 v_i ，根据线性代数知识我们知道有公式： $Sv_i = \lambda_i v_i, i = 1, 2, \dots, n$

4. 对特征值按照大小进行递减排序，特征向量的顺序和特征值是一致的。假设我们只需保留 K 个维数 ($K < n$)，则我们会选取特征值最大的前 K 个特征向量，用这 K 个特征向量，来表示图像，这 K 个向量就是图像 K 个主成分分量。

Opencv 里可以调用 PCA `pca(NewVector, Mat(), CV_PCA_DATA_AS_ROW, K)` 来直接求解 PCA。得到了这个 `pca` 后，我们可以直接用 `pca.mean`, `pca.eigenvalues`, `pca.eigenvectors` 来调用的（前 K 大）特征向量均值、特征值、特征向量等信息，均是以 `Mat` 结构保存的。

注意，求均值、构协方差矩阵 `pca` 都会帮我们做；但是重建的时候要我们手动去操作。

2.5 PCA 降维的应用

假设 faces 这个 vector 储存了训练集里 0000~0099 所有人脸框的 Mat。现在我们要将他做一个 PCA 降维，并将结果保存起来。最初的时候 K 直接取 N，即保留所有特征值和特征向量。

```
PCA pca(NewVector, Mat(), CV_PCA_DATA_AS_ROW, K = N);
Mat mean = pca.mean.clone();
Mat_ <float> eigenvalues = pca.eigenvalues;
Mat eigenvectors = pca.eigenvectors.clone();

Mat sum = mean.clone();
for (int i = 0; i < 10; i++)
    sum = sum + eigenvectors.row(i) * 0.1;
Mat meanTenFace;
normalize((sum.reshape(1, Ylen)), meanTenFace, 0, 255, NORM_MINMAX, CV_8UC1);
imshow("Mean", meanTenFace);
```

以上代码构建了 PCA，并计算了前 10 张特征脸的平均值。

其中 N 表示训练的图片数，D 表示图片的总维度（长乘宽）。有一个要注意的问题是：本来的图片使用 CV_8U1 （8 位无符号整数）保存的，但是在做 PCA 运算的时候会涉及到浮点数的运算，所以我们要将其转成 CV_32FC1 （32 位浮点数）的格式。输出的时候要再转回来。

我们用 Mat_ 而不是 Mat 来储存特征值 eigenvalues，因为 Mat_ 支持直接用下标访问值。

当 K=10 时的平均特征图如右图所示。



题目还要求：根据能量来决定选取的维度。下列代码用来根据能量百分比选择特征向量个数。经过我测试发现，一般前几个维度所占的能量特别大，所以选取的维度不会很大（具体测试情况详见实验结果）。

```
double totalenergy = 0, nowenergy = 0;
double need = 1.0 * stoi(argv[1]) / 100.0;
for (int i = 0; i < K; i++)
    totalenergy += eigenvalues(i);
for (int i = 0; i < K; i++) {
    nowenergy += eigenvalues(i);
    if (nowenergy / totalenergy >= need) {
        K = i + 1;
        break;
    }
}
cout << "Due to energy " << stoi(argv[1]) << "%, " << K << " features will be chosen." << endl;
```

2.6 训练数据的保存和读取

当得到了 `eigenvectors` 后，我们要考虑最终要保存什么东西。我们需要一个降维的矩阵 `Transform`，以及之前 100 张图片作用了 `Transform` 后的结果（作用前矩阵太大，所以我们直接保存作用后的）。

以下代码就是来计算降维变换矩阵 `transEigenvectors`（它其实就是 PCA 算出来的特征向量矩阵的转置），以及前 100 张图片作用于它后的结果 `reduceVector`（只需一个矩阵乘法就搞定了）。

```
Mat newEigenvectors = eigenvectors(Range(0, K), Range::all());
Mat transEigenvectors;
transpose(newEigenvectors, transEigenvectors);
Mat reduceVector = NewVector * transEigenvectors;
```

我搜了好久有关 `Mat` 的存储，最后发现 `FileStorage` 这个类最方便。

我会把各种 `Mat` 类和变量以 `xml` 的形式进行储存和读取，具体的代码见下。

储存（`Mytrain`）：

```
FileStorage fs(argv[2], FileStorage::WRITE);
fs << "N" << N;
fs << "K" << K;
fs << "transEigenvectors" << transEigenvectors;
fs << "reduceVector" << reduceVector;
fs << "mean" << mean;
fs.release();
```

读取（`Mytest`）：

```
FileStorage fs(argv[3], FileStorage::READ);
Mat transEigenvectors, reduceVector;
fs["N"] >> N;
fs["K"] >> K;
fs["reduceVector"] >> reduceVector;
fs["transEigenvectors"] >> transEigenvectors;
fs["mean"] >> transEigenvectors;
```

2.7 识别程序

识别程序的逻辑很简单。我们把用户指定的图片和眼睛数据给读入（具体细节我隐藏在 `readMat` 里了），然后将它带入降维变换矩阵，得出变换后的 `K` 维向量 `testVector`。随后我们遍历之前训练集里的 100 张图片，用欧几里得距离去判定，找到距离最近的一个图片向量，即为我们想要的结果。

```
testVector = testVector * transEigenvectors;
float bestdist = 1e20; int id = 0;
for (int i = 0; i < N; i++) {
    float dist = norm(reduceVector.row(i), testVector.row(0), NORM_L2);
    if (dist < bestdist) bestdist = dist, id = i;
    printf("Similarity with [%d] : %.2f\n", i, dist);
}
```


三、成果展示

```
C:\Windows\system32\cmd.exe
Microsoft Windows [版本 10.0.18362.476]
(c) 2019 Microsoft Corporation. 保留所有权利。





E:\ZJU大学生活\课程学习\计算机视觉\lab3\lab1>mytrain
Usage: mytrain.exe --energy% --model_file [--train_numbers]

E:\ZJU大学生活\课程学习\计算机视觉\lab3\lab1>mytrain 92 save.xml 100
Due to energy 92%, 34 features will be chosen.

E:\ZJU大学生活\课程学习\计算机视觉\lab3\lab1>mytest
Usage: mytest.exe --picture_file --eye_file --model_file

E:\ZJU大学生活\课程学习\计算机视觉\lab3\lab1>mytest mytest.pgm mytest.eye save.xml
```

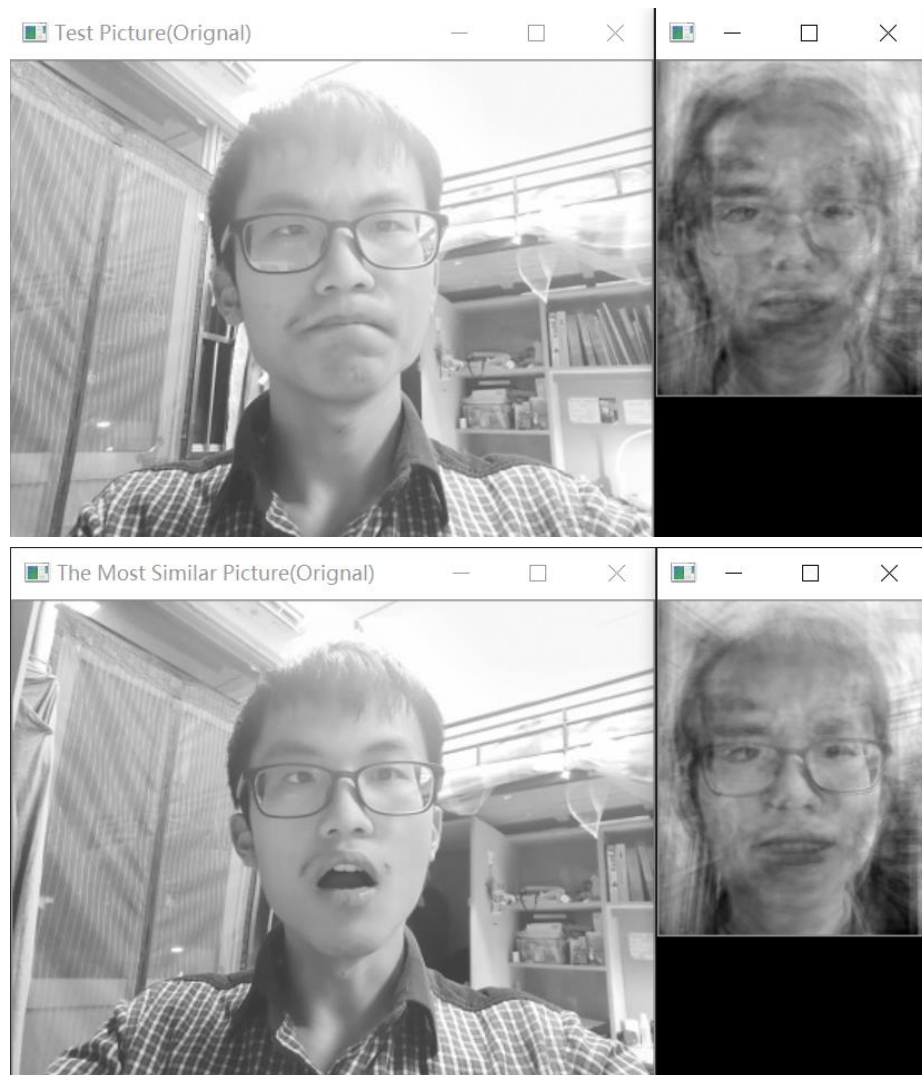
命令行测试

能量百分比	向量个数	原头像	重建后头像
98%	51/100		
90%	28/100		
60%	4/100		

特征向量重建测试



前 10 维特征的均值图片



测试结果（上面是测试图，下面是训练图，左侧是原图，右侧是 92% 能量的特征图）

另：我的数据集有 100 张图，大概 20 个人。我在剩余图片中把每个人的图片都测试了一遍，在较高能量度的情况下，检测到同人的正确率有 100%。但是对于同人不同表情的照片（面无表情 or 微笑 or 皱眉）检查结果不是很理想，有小部分测试数据匹配到了不同表情的同人。

```

Similarity with [21] : 10976.66
Similarity with [22] : 18512.65
Similarity with [23] : 18593.14
Similarity with [24] : 13649.95
Similarity with [25] : 16587.21
Similarity with [26] : 12647.27
Similarity with [27] : 14716.17
Similarity with [28] : 11435.87
Similarity with [29] : 14429.04
Similarity with [30] : 5569.30
Similarity with [31] : 5616.54
Similarity with [32] : 5315.98
Similarity with [33] : 13648.92
Similarity with [34] : 11189.53
Similarity with [35] : 17827.01
Similarity with [36] : 11688.01
Similarity with [37] : 12549.40
Similarity with [38] : 15768.32
Similarity with [39] : 15207.62
Similarity with [40] : 16661.62
Similarity with [41] : 11731.21
Similarity with [42] : 23199.87

```

欧几里得距离（30~32 是自己的图，可见匹配效果还是很不错的）

四、感想

本次实验看似步骤清晰，但是操作起来特别麻烦——。 “纸上得来终觉浅，绝知此事要躬行”，做实验+写报告前前后后加起来花了我八九个小时。

我感觉 C++ 的 opencv 搞得不太好，很多函数都很杂乱，不同函数之间参数传递可能也是混乱的（比如有些地方先传长再传宽，有些地方反过来）。最让人头疼的是，无论遇到什么 opencv 相关的错误，运行完都抛出一个奇怪的运行时错误（类似于 xxxx 地址内存访问异常），根本不给人调式的机会。我只能输出调试定位到错误行，再仔细检查 Mat 哪里用错了。

本实验还会经常有一些“小需求”，比如“把数据集整理好，打乱一下”、“安排各种文件的名称和路径”、“标注眼睛”、“暂时读取一下 pgm 图片看看是怎样的一张图（普通软件打不开 pgm）”，很多都需要再用 opencv 写一个小程序。而 C++ 写起程序来又特别累，所以特别耗时。

还有一个让我花时间的地方就是，opencv 函数太多太杂，有些函数一时间百度不到，文档查起来又特别累。每当我使用错误，程序马上给我跑奇怪的 RE，搞得人心态崩溃。举两个很经典的例子：

①我想访问 Mat 类里的成员变量，访问起来特别麻烦。RE 了很久才发现：可以定义 Mat_类，支持快速用下标访问 Mat 的成员。

②带入某些函数运算的时候，需要保证 Mat 类是连续储存的。经过某些函数操作后，它的地址可能变得不连续的。于是在传入新函数前，我做一步 `B=A.clone()`，这样即使 A 不连续存储，B 也会连续存储。

最开始做这个实验时，我没有做重建图片的测试。后来和同学讨论时，我突然发现这也是一个不错的尝试，就去试了一下——结果发现重建结果要不是全白，要不是极其模糊的人脸。于是我往回检查，才发现我对 pca 这个过程还不是很熟悉：PCA 函数会把传入的 $N \times D$ 的矩阵减去均值，求协方差矩阵，再计算前 K 大特征值。这些步骤都是封在 PCA 内部的。但是最终显示重建效果时需要我们手动操作：除了乘上坐标转换矩阵 $[D, K]$ ，还要加上均值。加上了均值后，重建就正确了。