CV 作业四报告

姓名：秦昇　学号：3120000060　课程：计算机视觉

软件开发说明：

1. 基于 OpenCV 库和 at&t 人脸数据库，输入一组 40 人每人固定 2-10 张标记了编号的人脸，分别用 Eigenface 和 Fisherface 进行识别和分解，得到对应的特征向量，构建出人脸识别系统。
2. 输入一定量的测试图片，分别用相同方法进行分解，判断其属于哪一编号的人脸图像，并计算标定准确度，通过作图等方法，进行算法间的对比分析。
3. 输出该图像对应的特征向量的主要部分系数，如果匹配正确，则在图像上显示相应信息。

算法具体步骤

1. **输入图像预处理**

首先拿到实验所需数据库——放在 attr_faces 文件夹中。由于数据量较大，为了便于分析，我们将图片路径以及对应编号按照一定格式放在一个 CSV 文件中，具体处理方式如下：

首先将所有图片路径输入文件 ，利用命令行参数（假定 attr_faces 文件位于 C:\）

```
C:\attr_faces>dir /b/s *.pgm > at.csv
```

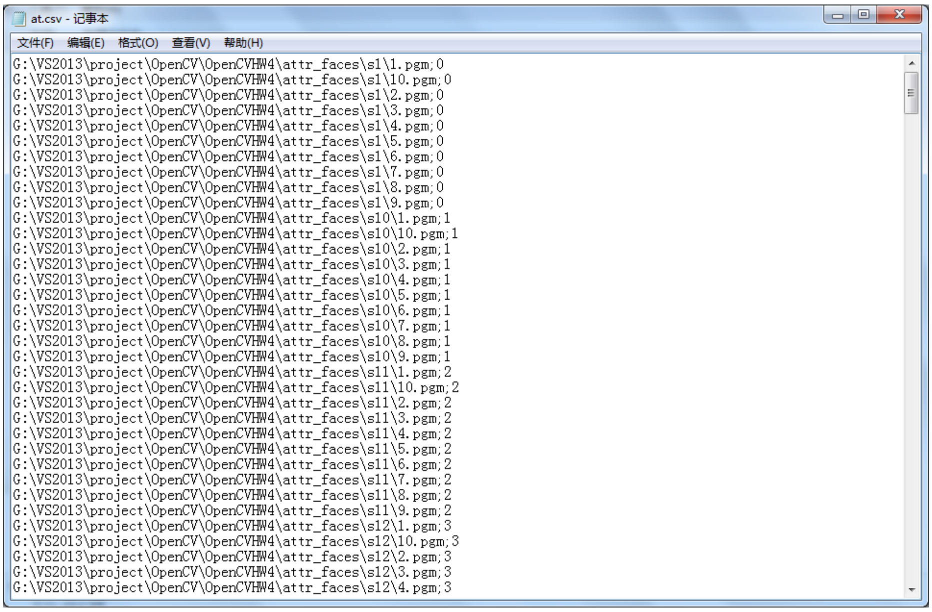
然后按照顺序进行标号，这里我使用了一个 Python 的小脚本

```
import sys
f = open("at.txt")
line = f.readline()
count = 0
num = 0
while line:
    line = line.strip('\n')
    sys.stdout.write(line)
    sys.stdout.write(';')
    print num
    count = count + 1
    if count % 10 == 0:
        num = num + 1

    line = f.readline()

f.close()
```

得到如下形式的结果 at.csv




```
G:\VS2013\project\OpenCV\OpenCVHW4\attr_faces\s1\1.pgm;0
G:\VS2013\project\OpenCV\OpenCVHW4\attr_faces\s1\10.pgm;0
G:\VS2013\project\OpenCV\OpenCVHW4\attr_faces\s1\2.pgm;0
G:\VS2013\project\OpenCV\OpenCVHW4\attr_faces\s1\3.pgm;0
G:\VS2013\project\OpenCV\OpenCVHW4\attr_faces\s1\4.pgm;0
G:\VS2013\project\OpenCV\OpenCVHW4\attr_faces\s1\5.pgm;0
G:\VS2013\project\OpenCV\OpenCVHW4\attr_faces\s1\6.pgm;0
G:\VS2013\project\OpenCV\OpenCVHW4\attr_faces\s1\7.pgm;0
G:\VS2013\project\OpenCV\OpenCVHW4\attr_faces\s1\8.pgm;0
G:\VS2013\project\OpenCV\OpenCVHW4\attr_faces\s1\9.pgm;0
G:\VS2013\project\OpenCV\OpenCVHW4\attr_faces\s10\1.pgm;1
G:\VS2013\project\OpenCV\OpenCVHW4\attr_faces\s10\10.pgm;1
G:\VS2013\project\OpenCV\OpenCVHW4\attr_faces\s10\2.pgm;1
G:\VS2013\project\OpenCV\OpenCVHW4\attr_faces\s10\3.pgm;1
G:\VS2013\project\OpenCV\OpenCVHW4\attr_faces\s10\4.pgm;1
G:\VS2013\project\OpenCV\OpenCVHW4\attr_faces\s10\5.pgm;1
G:\VS2013\project\OpenCV\OpenCVHW4\attr_faces\s10\6.pgm;1
G:\VS2013\project\OpenCV\OpenCVHW4\attr_faces\s10\7.pgm;1
G:\VS2013\project\OpenCV\OpenCVHW4\attr_faces\s10\8.pgm;1
G:\VS2013\project\OpenCV\OpenCVHW4\attr_faces\s10\9.pgm;1
G:\VS2013\project\OpenCV\OpenCVHW4\attr_faces\s11\1.pgm;2
G:\VS2013\project\OpenCV\OpenCVHW4\attr_faces\s11\10.pgm;2
G:\VS2013\project\OpenCV\OpenCVHW4\attr_faces\s11\2.pgm;2
G:\VS2013\project\OpenCV\OpenCVHW4\attr_faces\s11\3.pgm;2
G:\VS2013\project\OpenCV\OpenCVHW4\attr_faces\s11\4.pgm;2
G:\VS2013\project\OpenCV\OpenCVHW4\attr_faces\s11\5.pgm;2
G:\VS2013\project\OpenCV\OpenCVHW4\attr_faces\s11\6.pgm;2
G:\VS2013\project\OpenCV\OpenCVHW4\attr_faces\s11\7.pgm;2
G:\VS2013\project\OpenCV\OpenCVHW4\attr_faces\s11\8.pgm;2
G:\VS2013\project\OpenCV\OpenCVHW4\attr_faces\s11\9.pgm;2
G:\VS2013\project\OpenCV\OpenCVHW4\attr_faces\s12\1.pgm;3
G:\VS2013\project\OpenCV\OpenCVHW4\attr_faces\s12\10.pgm;3
G:\VS2013\project\OpenCV\OpenCVHW4\attr_faces\s12\2.pgm;3
G:\VS2013\project\OpenCV\OpenCVHW4\attr_faces\s12\3.pgm;3
G:\VS2013\project\OpenCV\OpenCVHW4\attr_faces\s12\4.pgm;3
```

2. 读入图片和标记并保存

虽然本次实验采用了 Eigenface 和 Fisherface 两种人脸识别的方法，但图片和标号的方式相同，采用如下静态方法：

```
static void read_csv(const string& filename, vector<Mat>& images,
                        vector<int>&   labels,   int   num,   char
separator = ';') {
       std::ifstream file(filename.c_str(), ifstream::in);
       if (!file) {
           string error_message = "No valid input file was given,
please check the given filename.";
           CV_Error(CV_StsBadArg, error_message);
       }
       string line, path, classlabel;
       int count = -1;
       while (getline(file, line)) {
           count++;
           if (count % 10 < num){
               stringstream liness(line);
               getline(liness, path, separator);
               getline(liness, classlabel);
               if (!path.empty() && !classlabel.empty()) {
                   images.push_back(imread(path, 0));
                   labels.push_back(atoi(classlabel.c_str()));
               }
           }
       }
```

```
}
```

将读入的图片存入 images 和 labels 两个矩阵中。这里加入 num，这是用来决定每一个人的训练集的人脸数（每人最多 10 张，9 张作训练集，1 张进行测试）。

3. Eigenface 的训练和测试

在得到图片集和标号集后，用 Eigenface 来检测，仅仅如下代码：

```
Ptr<FaceRecognizer> model = createEigenFaceRecognizer();
model->train(images, labels);
```

两个函数即可。这里我们将每一个人的第 num 张图片拿出来，加入测试集，并从训练集中删除。测试阶段，对每一张测试图片，若测试所得标号与实际标号相同，则说明测试结果正确，在图片上输出正确信息和对应的编号，并将正确计数器增加。

这里我们还将 Eigenface 算法中，PCA 向量分解算法里前 10th 主分量的系数输出。

在所有测试完成后，根据正确的数量 right 比测试集总数 40，即可得到当前算法和训练集数目情况下，人脸识别的大致准确率。完整代码如下

```
void Eigen_Face(vector<Mat> images, vector<int> labels,string
output_folder, int num){
        // 得到第一张照片的高度. 在下面对图像
        // 变形到他们原始大小时需要
        cout << "Eigenface!" << endl;
        int height = images[0].rows;

        vector<Mat> testSample;
        vector<int> testLabel;
        for (int i = num - 1; i < num*40; i += num){
            testSample.push_back(images[i]);
            testLabel.push_back(labels[i]);
        }
        for (int i = num * 40 - 1; i > 0; i -= num){
            images.erase(images.begin() + i);
```

```
            labels.erase(labels.begin() + i);
        }
        // 下面几行创建了一个特征脸模型用于人脸识别，

        Ptr<FaceRecognizer> model = createEigenFaceRecognizer();
        model->train(images, labels);
        int right = 0;
        // 下面对测试图像进行预测，predictedLabel 是预测标签结果
        for (int i = 0; i < testSample.size(); i++){
            int predictedLabel = model->predict(testSample[i]);
            string result_message = format("Predicted class %d
= %d / Actual class%d = %d.",i, predictedLabel, i,testLabel[i]);
            Mat img(testSample[i]);
            if (predictedLabel == testLabel[i]){
                right++;
                char no[5];
                _itoa(i, no, 10);
                string n(no);
                string output = "Right!No." + n;
                putText(img, output, Point(10, 10),
CV_FONT_HERSHEY_COMPLEX,
                    0.4, Scalar(255, 255, 255));
            }
            imshow("test", img);
            cout << result_message << endl;
            // 这里是如何获取特征脸模型的特征值的例子，使用了 getMat 方法：
            Mat eigenvalues = model->getMat("eigenvalues");
            // 同样可以获取特征向量：
            Mat W = model->getMat("eigenvectors");
```

```
            // 得到训练图像的均值向量
            Mat mean = model->getMat("mean");

            // 现实还是保存特征脸:
            for (int i = 0; i < min(10, W.cols); i++) {
                string msg = format("Eigenvalue #%d = %.5f", i,
eigenvalues.at<double>(i));
                cout << msg << endl;
            }

            waitKey();
        }
        // 如果我们不是存放到文件中，就显示他，这里使用了暂定等待键盘输入:
        cout << "Accuracy = " << (float)right / 40 << endl;
}
```

4. Fisherface 的训练和测试

Fisherface 的实现过程和 Eigenface 几乎完全相同，只是这里输出的是利用 LDA（Linear Discriminant Analysis，线性判别分析）来进行分解，然而当单个人训练集较少的时候，其离散度远大于 Eigenface 中使用的 PCA 算法，前 16 个分量大多数为 0；所以我们不输出。

算法实现要点

1. **Eigenface 算法** （参考 Turk M, Pentland A. Eigenfaces for recognition[J]. Journal of cognitive neuroscience, 1991, 3(1): 71-86.

参考 http://www.tuicool.com/articles/umauie）

特征脸(Eigenface)理论基础-PCA(主成分分析法) 。

步骤一：获取包含 M 张人脸图像的集合 S。在我们的例子里有 25 张人脸图像（虽然是 25 个不同人的人脸的图像，但是看着怎么不像呢，难道我有脸盲症么），如下图所示哦。每张图像可以转换成一个 N 维的向量（是的，没错，一个像素一个像素的排成一行就好了，至于是横着还是竖着获取原图像的像素，随你自己，只要前后统一就可以），然后把这 M 个向量放到一个集合 S 里，如下式所示。

$$S = \{ \Gamma_1, \Gamma_2, \Gamma_3, \ .... \ , \Gamma_M \}$$

步骤二：在获取到人脸向量集合 S 后，计算得到平均图像 Ψ，至于怎么计算平均图像，公式在下面。就是把集合 S 里面的向量遍历一遍进行累加，然后取平均值。得到的这个 Ψ 其实还挺有意思的，Ψ 其实也是一个 N 维向量，如果再把它还原回图像的形式的话，可以得到如下的“平均脸”，是的没错，还他妈的挺帅啊。那如果你想看一下某计算机学院男生平均下来都长得什么样子，用上面的方法就可以了。

$$\Psi = \frac{1}{M} \sum_{n=1}^{M} \Gamma_n$$

步骤三：计算每张图像和平均图像的差值 Φ ，就是用 S 集合里的每个元素减去步骤二中的平均值。

$$\Phi_i = \Gamma_i - \Psi$$

步骤四：找到 M 个正交的单位向量 un，这些单位向量其实是用来描述 Φ （步骤三中的差值）分布的。un 里面的第 k（k=1,2,3...M)个向量 uk 是通过下式计算的，

$$\lambda_k = \frac{1}{M}\sum_{n=1}^{M}\left(u_k^T \Phi_n\right)^2$$

当这个 λ k（原文里取了个名字叫特征值）取最小的值时，uk 基本就确定了。补充一下，刚才也说了，这 M 个向量是相互正交而且是单位长度的，所以啦，uk 还要满足下式：

$$u_l^T u_k = \delta_{lk} = \begin{cases} 1 & \text{if } l = k \\ 0 & \text{otherwise} \end{cases}$$

上面的等式使得 uk 为单位正交向量。计算上面的 uk 其实就是计算如下协方差矩阵的特征向量：

$$C = \frac{1}{M}\sum_{n=1}^{M}\Phi_n\Phi_n^T$$
$$= AA^T$$

其中

$$A = \{\Phi_1, \Phi_2, \Phi_3, \ldots\ldots, \Phi_n\}$$

对于一个 NxN（比如 100x100）维的图像来说，上述直接计算其特征向量计算量实在是太大了（协方差矩阵可以达到 10000x10000），所以有了如下的简单计算。

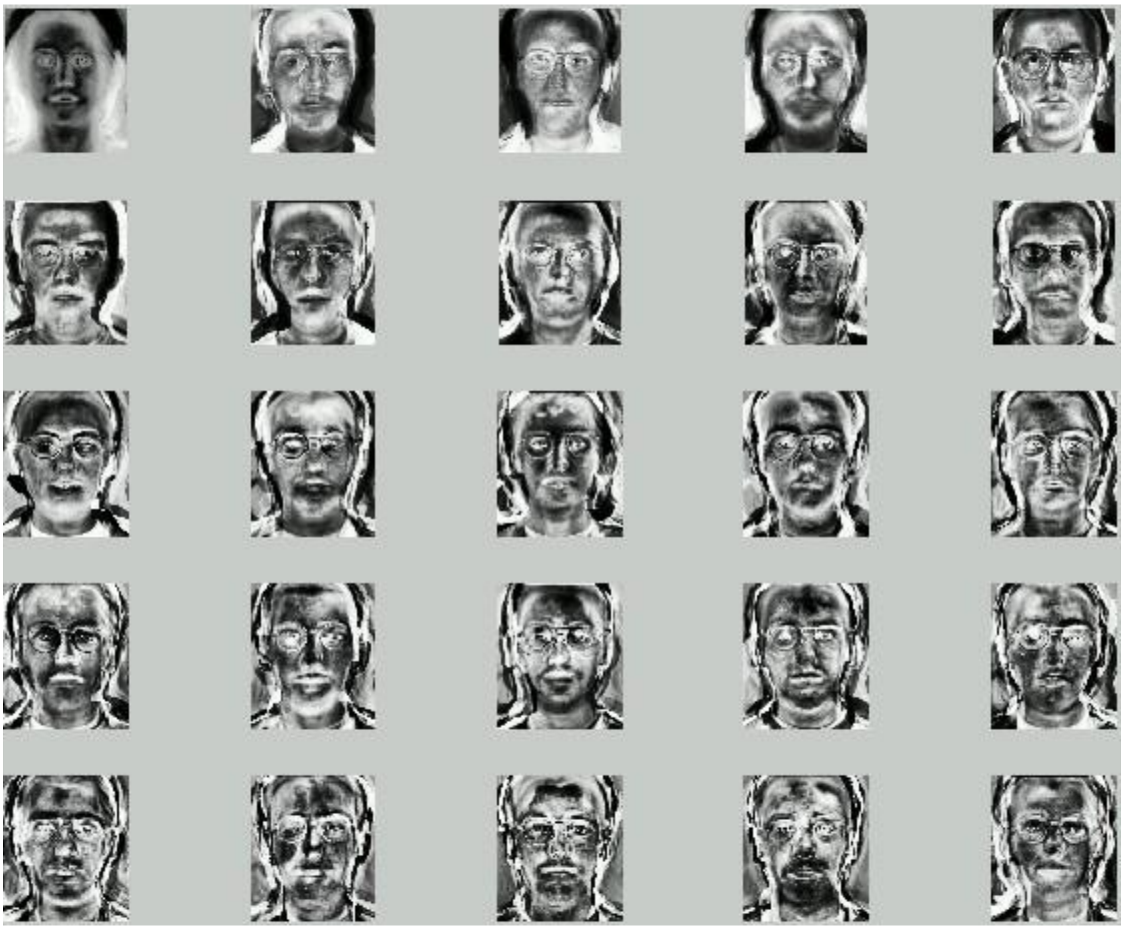
步骤四另解：如果训练图像的数量小于图像的维数比如（M<N^2)，那么起作用的特征向量只有 M-1 个而不是 N^2 个（因为其他的特征向量对应的特征值为 0），所以求解特征向量我们只需要求解一个 NxN 的矩阵。这个矩阵就是步骤四中的 AAT，我们可以设该矩阵为 L，那么 L 的第 m 行 n 列的元素可以表示为：

$$L_{mn} = \Phi_m^T \Phi_n$$

一旦我们找到了 L 矩阵的 M 个特征向量 **vl**，那么协方差矩阵的特征向量 **ul** 就可以表示为：

$$u_l = \sum_{k=1}^{M} v_{lk} \Phi_k \qquad l = 1, \ldots\ldots, M$$

这些特征向量如果还原成像素排列的话，其实还蛮像人脸的，所以称之为特征脸（如下图）。图里有二十五个特征脸，数量上和训练图像相等只是巧合。有论文表明一般的应用 40 个特征脸已经足够了。论文 Eigenface for recognition 里只用了 7 个特征脸来表明实验。



步骤五：识别人脸。OK，终于到这步了，别绕晕啦，上面几步是为了对人脸进行降维找到表征人脸的合适向量的。首先考虑一张新的人脸，我们可以用特征脸对其进行标示：

$$\omega_k = u_k^T(\Gamma - \Psi)$$

其中 k=1,2...M,对于第 k 个特征脸 uk，上式可以计算其对应的权重，M 个权重可以构成一个向量：

$$\Omega^T = [\omega_1, \omega_2, \ \ldots\ldots, \omega_M]$$

perfect，这就是求得的特征脸对人脸的表示了！

那如何对人脸进行识别呢，看下式：

$$\varepsilon_k = \left\|\Omega - \Omega_k\right\|^2$$

其中Ω代表要判别的人脸，Ωk 代表训练集内的某个人脸，两者都是通过特征脸的权重来表示的。式子是对两者求欧式距离，当距离小于阈值时说明要判别的脸和训练集内的第 k 个脸是同一个人的。当遍历所有训练集都大于阈值时，根据距离值的大小又可分为是新的人脸或者不是人脸的两种情况。根据训练集的不同，阈值设定并不是固定的。

2. **Fisherface 算法**（参考 Belhumeur P N, Hespanha J P, Kriegman D. Eigenfaces vs. fisherfaces: Recognition using class specific linear projection[J]. Pattern Analysis and Machine Intelligence, IEEE Transactions on, 1997, 19(7): 711-720.

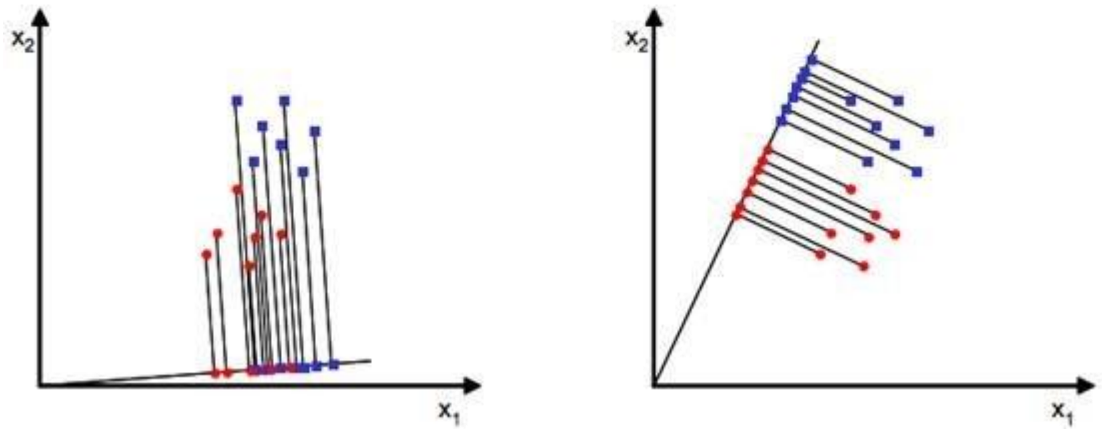
参考 http://blog.csdn.net/feirose/article/details/39552887）

Fisherface 所基于的 LDA（Linear Discriminant Analysis，线性判别分析）理论和 特征脸 里用到的 PCA 有相似之处，都是对原有数据进行整体降维映射到低维空间的方法，LDA 和 PCA 都是从数据整体入手而不同于 LBP 提取局部纹理特征。如果阅读本文有难度，可以考虑自学斯坦福公开课机器学习或者补充线代等数学知识。

同时作者要感谢 cnblogs 上的大牛 JerryLead，本篇博文基本摘自他的线性判别分析（Linear Discriminant Analysis）[1]。

1、数据集是二类情况

通常情况下，待匹配人脸要和人脸库内的多张人脸匹配，所以这是一个多分类的情况。出于简单考虑，可以先介绍二类的情况然后拓展到多类。假设有二维平面上的两个点集 x（x 是包含横纵坐标的二维向量），它们的分布如下图（1）（分别以蓝点和红点表示数据）：



原有数据是散布在平面上的二维数据，如果想用一维的量（比如到圆点的距离）来合理的表示而且区分开这些数据，该怎么办呢？一种有效的方法是找到一个合适的向量 w（和数据相同维数），将数据投影到 w 上（会得到一个标量，直观的理解就是投影点到坐标原点的距离），根据投影点来表示和区分原有数据。以数学公式给出投影点到到原点的距离： y=w T x。图（1）给出了两种 w 方案，w 以从原点出发的直线来表示，直线上的点是原数据的投影点。直观判断右侧的 w 更好些，其上的投影点能够合理的区分原有的两个数据集。但是计算机不知道这些，所以必须要有确定的方法来计算这个 w。

首先计算每类数据的均值（中心点）：

$$\mu_i = \frac{1}{N_i}\sum_{x\in\omega_i} x$$

这里的 i 是数据的分类个数，Ni 代表某个分类下的数据点数，比如 u1 代表红点的中心，u2 代表蓝点的中心。

数据点投影到 w 上的中心为：

$$\widetilde{\mu_i} = \frac{1}{N_i}\sum_{y\in\omega_i} y = \frac{1}{N_i}\sum_{x\in\omega_i} w^T x = w^T \mu_i$$

如何判断向量 w 最佳呢，可以从两方面考虑：1、 不同的分类得到的投影点要尽量分开；2、同一个分类 投影后得到的点要尽量聚合 。从这两方面考虑，可以定义如下公式：

$$J(w) = |\widetilde{\mu_1} - \widetilde{\mu_2}| = |w^T(\mu_1 - \mu_2)|$$

J(w)代表不同分类投影中心的距离，它的值越大越好。

$$\tilde{s_i}^2 = \sum_{y \in \omega_i} (y - \tilde{\mu}_i)^2$$

上式称之为散列值（scatter matrixs），代表同一个分类投影后的散列值，也就是投影点的聚合度，它的值越小代表投影点越聚合。

结合两个公式，第一个公式做分子另一个做分母：

$$J(w) = \frac{|\widetilde{\mu_1} - \widetilde{\mu_2}|^2}{\tilde{s_1}^2 + \tilde{s_2}^2}$$

上式是 w 的函数，值越大 w 降维性能越好，所以下面的问题就是求解使上式取最大值的 w。

把散列函数展开：

$$\tilde{s_i}^2 = \sum_{y \in \omega_i} (y - \tilde{\mu}_i)^2 = \sum_{x \in \omega_i} (w^T x - w^T \mu_i)^2 = \sum_{x \in \omega_i} w^T (x - \mu_i)(x - \mu_i)^T w$$

可以发现除 w 和 w^T 外，剩余部分可以定义为：

$$S_i = \sum_{x \in \omega_i} (x - \mu_i)(x - \mu_i)^T$$

其实这就是原数据的散列矩阵了，对不对。对于固定的数据集来说，它的散列矩阵也是确定的。

另外定义：

$$S_w = S_1 + S_2$$

S w 称为 Within -class scatter matrix。

回到 $\tilde{s_i}^2$ 并用上面的两个定义做替换，得到：

$$\tilde{s_i}^2 = w^T S_i w$$

$$\tilde{s_1}^2 + \tilde{s_2}^2 = w^T S_w w$$

展开 J(w)的分子并定义 S B， S B 称为 Between-class scatter。

$$(\tilde{\mu}_1-\tilde{\mu}_2)^2=(w^T\mu_1-w^T\mu_2)^2=w^T\underbrace{(\mu_1-\mu_2)(\mu_1-\mu_2)^T}_{S_B}w=w^TS_Bw$$

这样就得到了 J(w)的最终表示：

$$J(w)=\frac{w^TS_Bw}{w^TS_ww}$$

上式求极大值可以利用 拉格朗日乘数法，不过需要限定一下分母的值，否则分子分母都变，怎么确定最好的 w 呢。可以令 $||w^TS_ww||=1$ ，利用拉格朗日乘数法得到：

$$c(w)=w^tS_Bw-\lambda(w^tS_Ww-1)$$
$$\Rightarrow\frac{dc}{dw}=2S_Bw-2\lambda S_Ww=0$$
$$\Rightarrow S_Bw=\lambda S_Ww$$

其中 w 是矩阵，所以求导时可以把 w^TS_ww 当做 S_ww^2 。（这点我也不懂）

上式两边同乘以 S_w^{-1} 可以得到：

$$S_W{}^{-1}S_Bw=\lambda w$$

可以发现 w 其实就是矩阵 $S_w^{-1}S_B$ 的特征向量了对不对。

通过上式求解 w 还是有些困难的，而且 w 会有多个解，考虑下式：

$$S_B=(\mu_1-\mu_2)(\mu_1-\mu_2)^T$$

将其带入下式：

$$S_Bw=(\mu_1-\mu_2)(\mu_1-\mu_2)^Tw=(\mu_1-\mu_2)*\lambda_w$$

其中 λw 是以 w 为变量的数值，因为(u1-u2)^T 和 w 是相同维数的，前者是行向量后者列向量。继续带入以前的公式：

$$S_w^{-1}S_Bw=S_w^{-1}(\mu_1-\mu_2)*\lambda_w=\lambda w$$

由于 w 扩大缩小任何倍不影响结果，所以可以约去两遍的未知常数 λ 和 λw （存疑）：

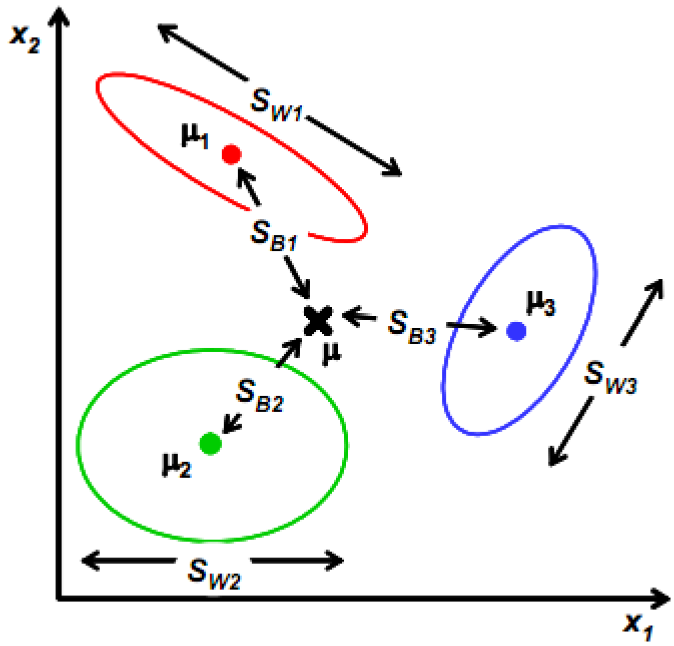
$$w = S_w^{-1}(\mu_1 - \mu_2)$$

到这里，w 就能够比较简单的求解了。

2、数据集是多类的情况

这部分是本博文的核心。假设有 C 个人的人脸图像，每个人可以有多张图像，所以按人来分，可以将图像分为 C 类，这节就是要解决如何判别这 C 个类的问题。判别之前需要先处理下图像，将每张图像按照逐行逐列的形式获取像素组成一个向量，和第一节类似设该向量为 x，设向量维数为 n，设 x 为列向量（n 行 1 列）。

和第一节简单的二维数据分类不同，这里的 n 有可能成千上万，比如 100x100 的图像得到的向量为 10000 维，所以第一节里将 x 投影到一个向量的方法可能不适用了，比如下图：



图（2）

平面内找不到一个合适的向量，能够将所有的数据投影到这个向量而且不同类间合理的分开。所以我们需要增加投影向量 w 的个数（当然每个向量维数和数据是相同的，不然怎么投影呢），设 w 为：

$$W = [w_1 | w_2 | \ldots | w_K]$$

w1、w2 等是 n 维的列向量，所以 w 是个 n 行 k 列的矩阵，这里的 k 其实可以按照需要随意选取，只要能合理表征原数据就好。x 在 w 上的投影可以表示为：

$$y = W^T x$$

所以这里的 y 是 k 维的列向量。

像上一节一样，我们将从投影后的类间散列度和类内散列度来考虑最优的 w，考虑图（2）中二维数据分为三个类别的情况。与第一节类似， μ i 依然代表类别 i 的中心，而 S w 定义如下：

$$S_w = \sum_{i=1}^{c} S_{wi}$$

其中：

$$S_{wi} = \sum_{x \in \omega_i} (x - \mu_i)(x - \mu_i)^T$$

代表类别 i 的类内散列度，它是一个 nxn 的矩阵。

所有 x 的中心 μ 定义为：

$$\mu = \frac{1}{N} \sum_{\forall x} x = \frac{1}{N} \sum_{x \in \omega_i} N_i \mu_i$$

类间散列度定义和上一节有较大不同：

$$S_B = \sum_{i=1}^{c} N_i (\mu_i - \mu)(\mu_i - \mu)^T$$

代表的是每个类别到 μ 距离的加和，注意 Ni 代表类别 i 内 x 的个数，也就是某个人的人脸图像个数。

上面的讨论都是投影之间的各种数据，而 J(w)的计算实际是依靠投影之后数据分布的，所以有：

$$\tilde{\mu}_i = \frac{1}{N_i} \sum_{y \in \omega_i} y$$

$$\tilde{\mu} = \frac{1}{N} \sum_{\forall y} y$$

$$\widetilde{S_w} = \sum_{i=1}^{C} \sum_{y \in \omega_i} (y - \widetilde{\mu_i})(y - \widetilde{\mu_i})^T$$

$$\widetilde{S_B} = \sum_{i=1}^{C} N_i(\widetilde{\mu_i} - \widetilde{\mu})(\widetilde{\mu_i} - \widetilde{\mu})^T$$

分别代表投影后的类别 i 的中心，所有数据的中心，类内散列矩阵，类间散列矩阵。与上节类似 J(w)可以定义为：

$$\widetilde{S_w} = W^T S_w W$$

$$\widetilde{S_B} = W^T S_B W$$

回想我们上节的公式 J(w)，分子是两类中心距，分母是每个类自己的散列度。现在投影方向是多维了（好几条直线），分子需要做一些改变，我们不是求两两样本中心距之和（这个对描述类别间的分散程度没有用），而是求每类中心相对于全样本中心的散列度之和。得到：

$$J(w) = \frac{|\widetilde{S_B}|}{|\widetilde{S_w}|} = \frac{|W^T S_B W|}{|W^T S_w W|}$$

最后化为：

$$S_w^{-1} S_B w_i = \lambda w_i$$

还是求解矩阵的特征向量，然后根据需求取前 k 个特征值最大的特征向量。

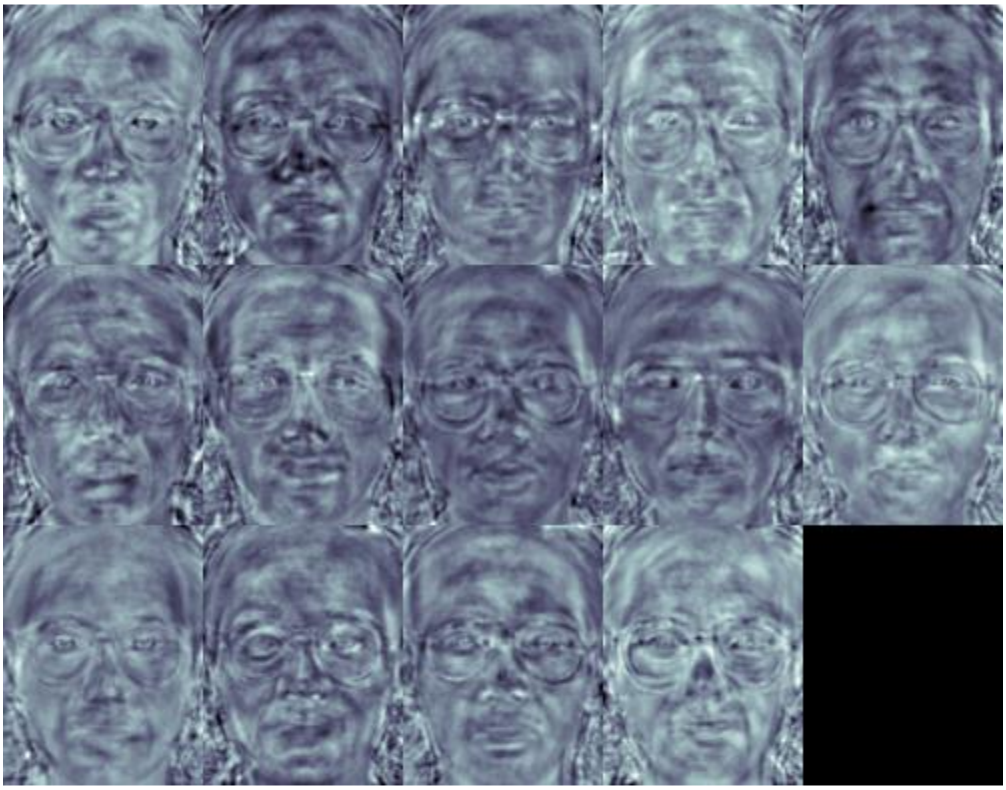
另外还需注意：

由于 S B 中的（μi-μ）秩为 1，所以 S B 的至多为 C（矩阵的秩小于等于各个相加矩阵的和）。又因为知道了前 C-1 个 μi 后，最后一个 μc 可以用前面的 μi 来线性表示，因此 S B 的秩至多为 C-1，所以矩阵的特征向量个数至多为 C-1。因为 C 是数据集的类别，所以假设有 N 个人的照片，那么至多可以取到 N-1 个特征向量来表征原数据。（存疑）

如果你读过前面的一篇文章 PCA 理论分析，会知道 PCA 里求得的特征向量都是正交的，但是这里的 $S_w^{-1} S_B$ 并不是对称的，所以求得的 K 个特征向量不一定正交，这是 LDA 和 PCA 最大的不同。

如前所述，如果在一个人脸集合上求得 k 个特征向量，还原为人脸图像的话就像下面

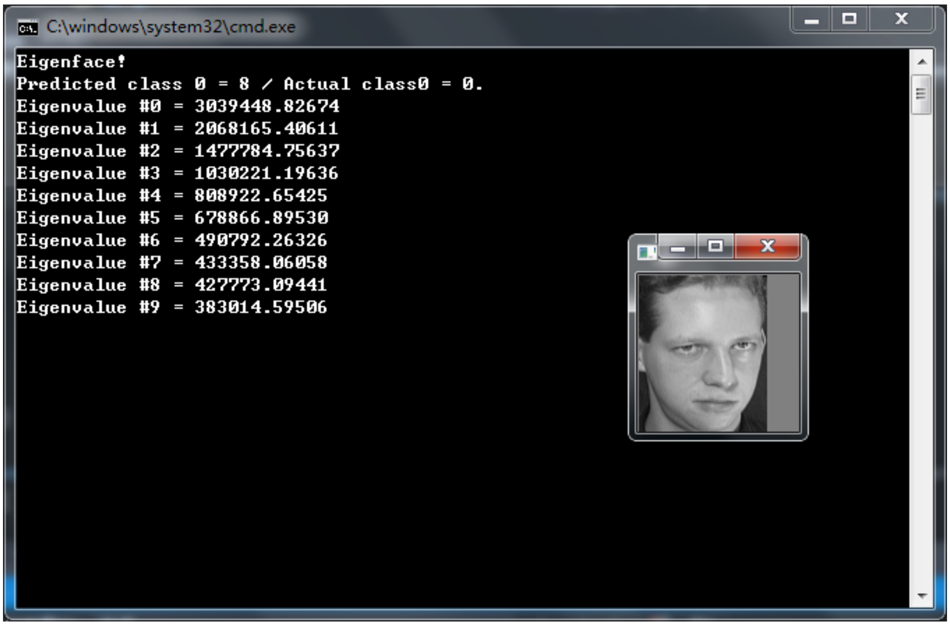
这样：



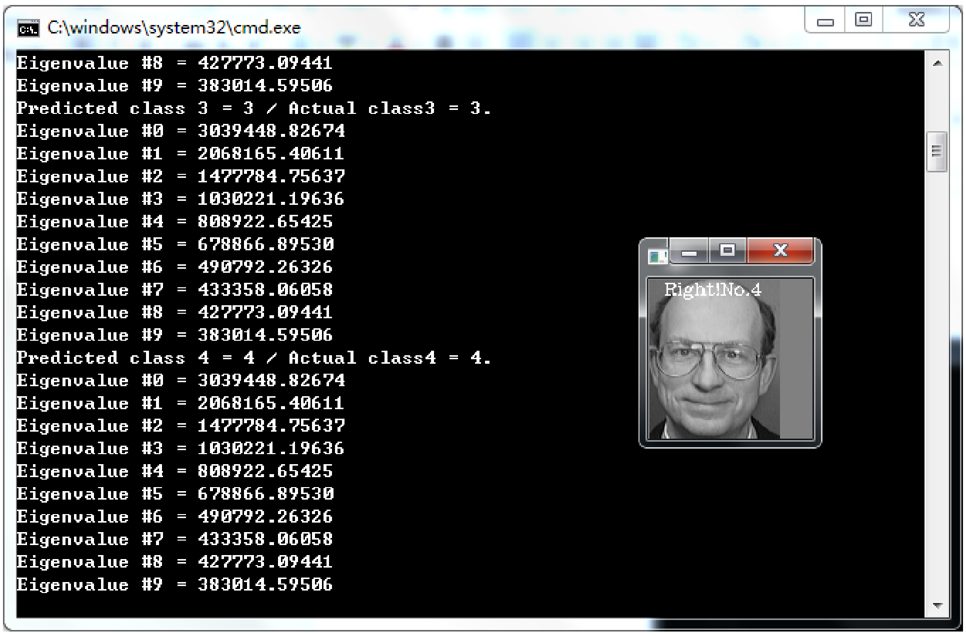
得到了 k 个特征向量，如何匹配某人脸和数据库内人脸是否相似呢，方法是将这个人脸在 k 个特征向量上做投影，得到 k 维的列向量或者行向量，然后和已有的投影求得欧式距离，根据阈值来判断是否匹配。

实验结果展示

1. Eigenface（num = 2 时即每个人只有一张图片作训练集）



正确匹配

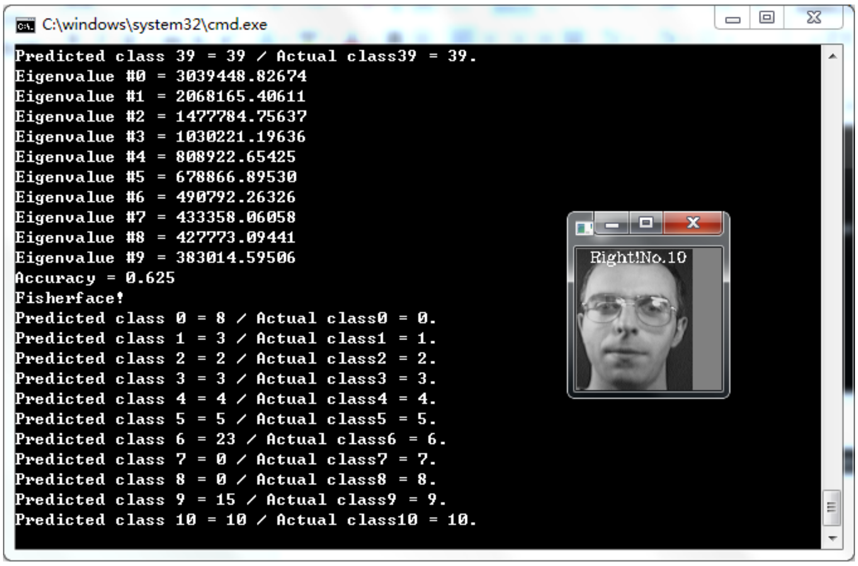


输出准确率： 

2. Fisherface（num = 2 时）

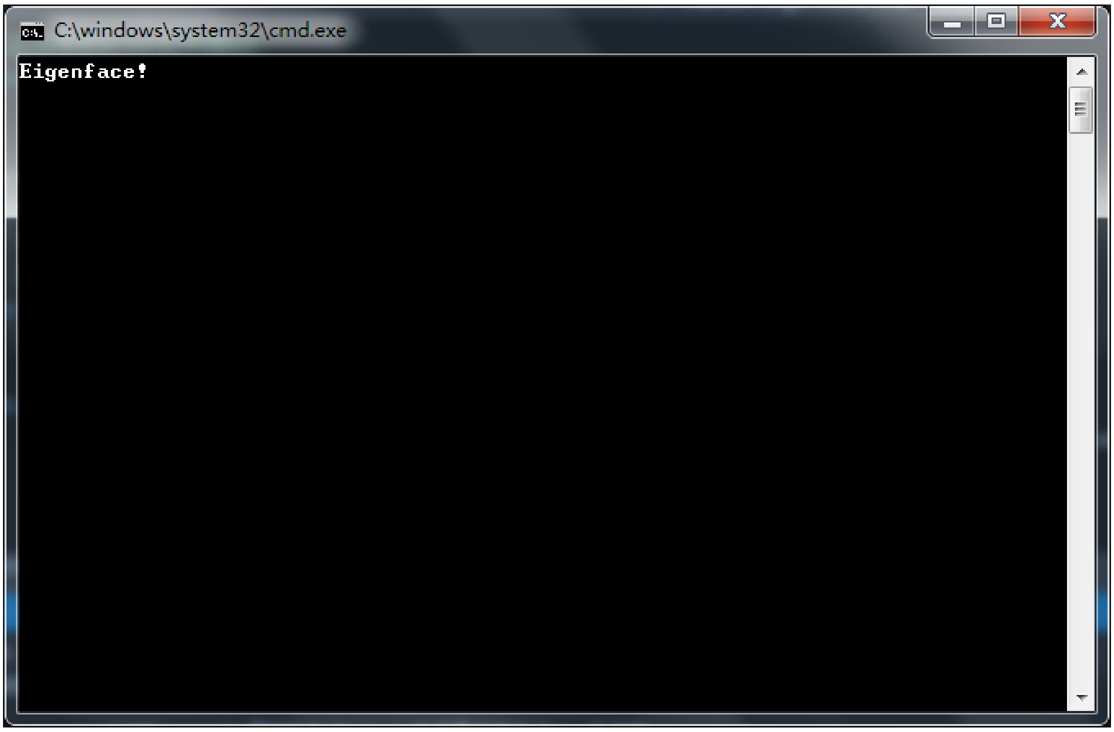
从 Fisherface！以下为其测试及输出

开始运行，用时明显较长

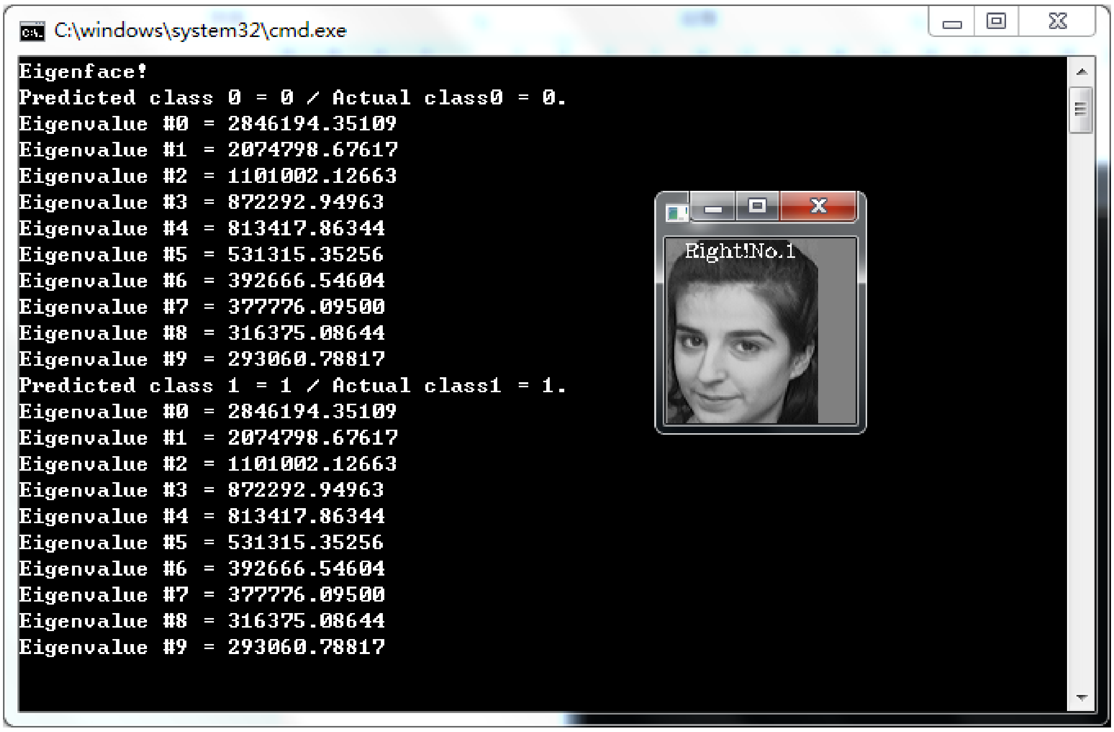


准确率：

3. Eigenface(num = 10 每个人有 9 张图片作训练集)

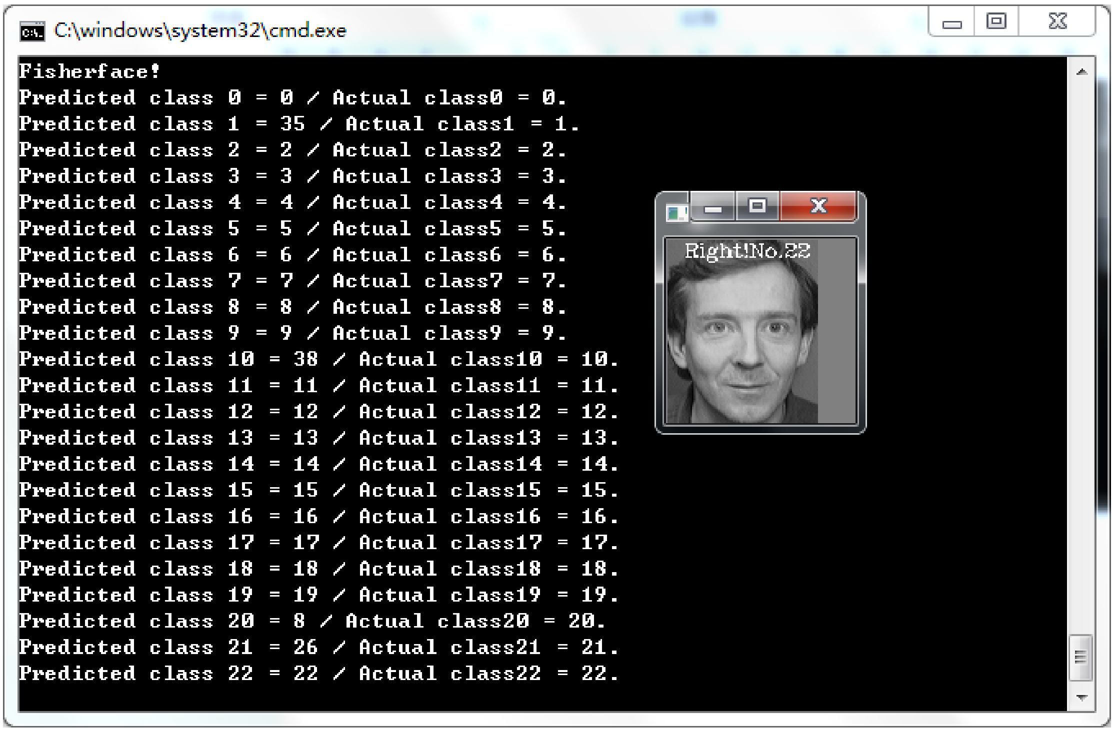


开始运行，用时明显较长



输出准确率：Accuracy = 0.975

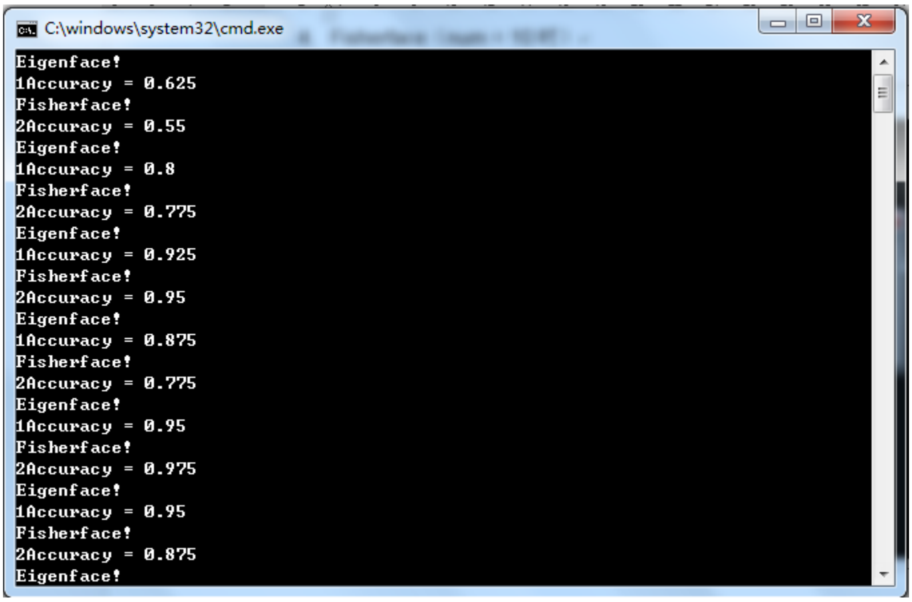
4. Fisherface（num = 10 时）

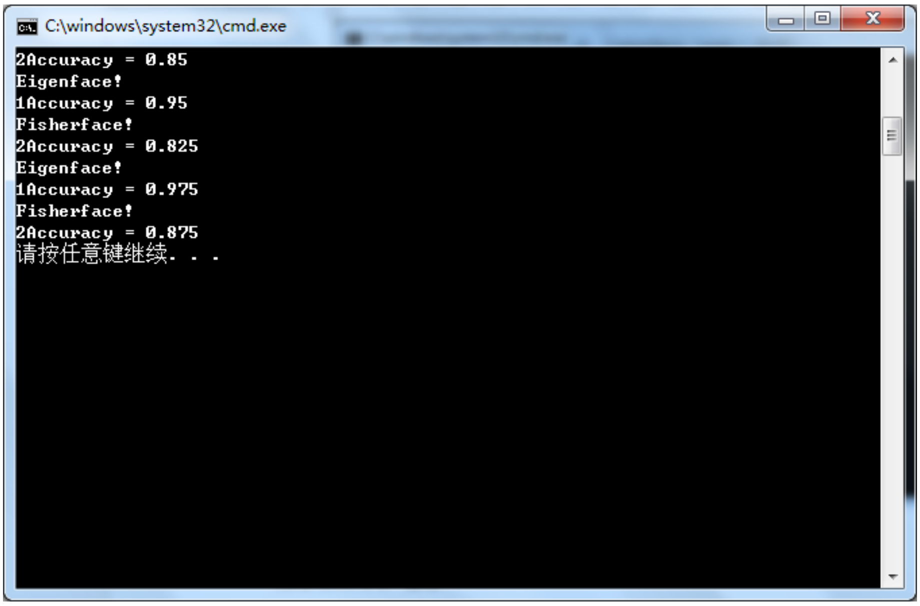


准确率：Accuracy = 0.875

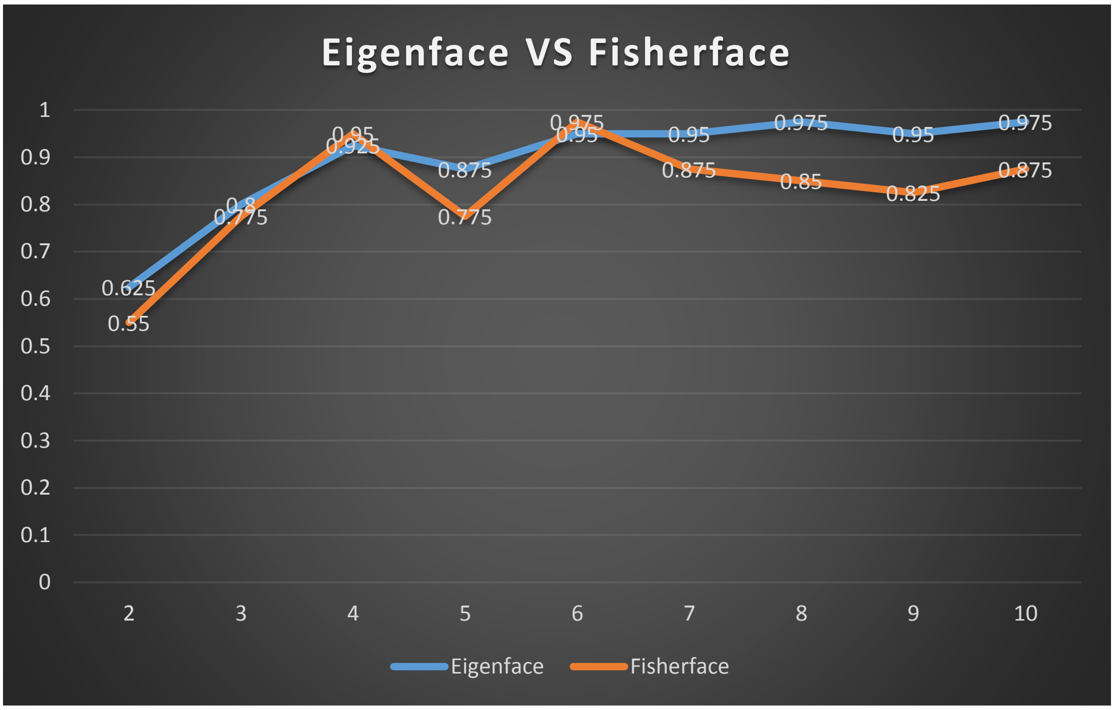
3. 两算法性能比较

首先修改代码，将 num 从 2-10 遍历，得出对应的准确度





这里我简单地进行了可视化



从图表可以看出，Eigenface 普遍准确率较 Fisherface 要更高一些，在训练集数量较小的情况下已经达到了近乎 100%的表现，但是 Fisherface 在 num = 4 和 num = 6 的时候比 Eigenface 略高。

编程体会

1. **Eigenface 和 Fisherface 算法的比较**

   本次试验对两个算法进行对比我主要针对人脸识别的准确度，得到了 Eigenface 的平均准确度较 Fisherface 要更高一些。但是也存在一些特殊点的情况如 num = 4 和 num = 6，原因有很多，比如数据集太少，噪点的影响很大等。

   此外，在 num = 10 的时候两个算法都出现了耗时明显的疲态，此时训练集图片数尚不足 400，这使我们对大量人脸图片的识别产生了质疑，也许算法仍然有待改进。

2. **适当地将数据可视化**

   在进行数据比较等工作的时候，可视化是一项十分有力也十分重要的手段，不仅可以帮助我们更加准确地分析数据，还可以让我们将数据更加直观和简单地呈献给我们的读者。