HW#4 Eigenface

作业要求：

写代码实现Eigenface人脸识别的训练、识别、重构过程：

1. 假设每张人脸图片中只有一张人脸，且两只眼睛位置已知(即：可人工标注给出)。例如：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在图像中的位置。
2. 实现三个程序过程，分别对应训练、识别、重构
3. 自己构建一个人脸库（至少40人，必须要含有自己的人脸图像），课程提供一个AT&T人脸库可选用
4. **不能直接调用OpenCV里面与Eigenface相关的函数**，特征值与特征向量求解可以调用SDK
5. **训练过程**，大致为："mytrain(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。演示程序同时将10个特征脸拼成一张图像，然后显示出来。
6. **识别过程**，大致为："mytest(输入人脸图像文件名 model文件名 其他参数...)", 将model文件装载进来后，对输入到人脸图像进行识别。演示程序将识别结果叠在输入到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。
7. **重构过程**，大致为："myreconstruct (输入人脸图像文件名 model文件名其他参数)"，将model文件装载进来后，对输入到人脸图像进行变换到特征脸空间，然后再用变换后到结果重构回原来人脸图像。演示程序可以同时显示用10个PCs、25个PCs、50个 PCs、以及100个PCs重构的结果，并实现对自己图像的重构。
8. 实验报告里必须包括以下实验结果：a.平均人脸与至少前10个特征脸；b. 自己人脸图像的10个PCs、25个PCs、50个PCs、以及100个PCs重构的结果；c. AT&T人脸库每人的一半数据做训练，另一半做测试，展示随着PC增加，Rank-1识别率的变换曲线(即横坐标为使用的PC的数量、纵坐标为Rank-1 rate的曲线)。

个人信息：

姓名：鲍奕帆

学号：3180103499

邮箱：3180103499@zju.edu.cn

个人画像：

完成日期：2020年12月28日 0时 28分。

软件开发说明

- 计算机与操作系统

macOS Catalina版本10.15.5

MacBook Pr(13-inch, 2020, Four Thunderbold 3 ports)

CPU: 2 GHz 四核 Intel Core i5

内存: 16 GB 3733 MHz LPDDR4

启动磁盘: Macintosh HD

图形卡: Intel Iris Plus Graphics 1536 MB

内核(uname -a):Darwin EvandeMacBook-Pro.local 19.5.0 Darwin Kernel Version 19.5.0: Tue May 26 20:41:44 PDT 2020; root:xnu-6153.121.2~2/RELEASE_X86_64 x86_64

- opencv版本

opencv_4.4.0

- 编程语言

C++11

- IDE

XCode Version 12.2(12B45b)

- 编译器

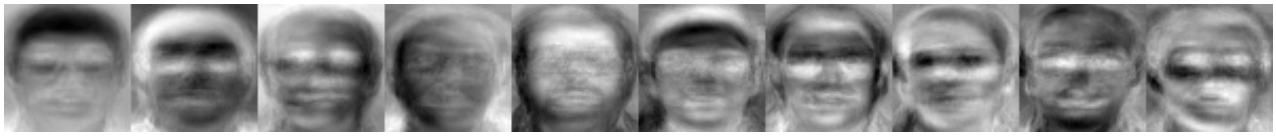
Apple clang version 11.0.3 (clang-1103.0.32.62)

- 字符编码

UTF-8

二、实验结果展示与分析

a. 平均人脸与前10个特征脸



b. 自己人脸图像的10个PCs、25个PCs、50个PCs、以及100个PCs重构的结果

10个PCs重构的结果

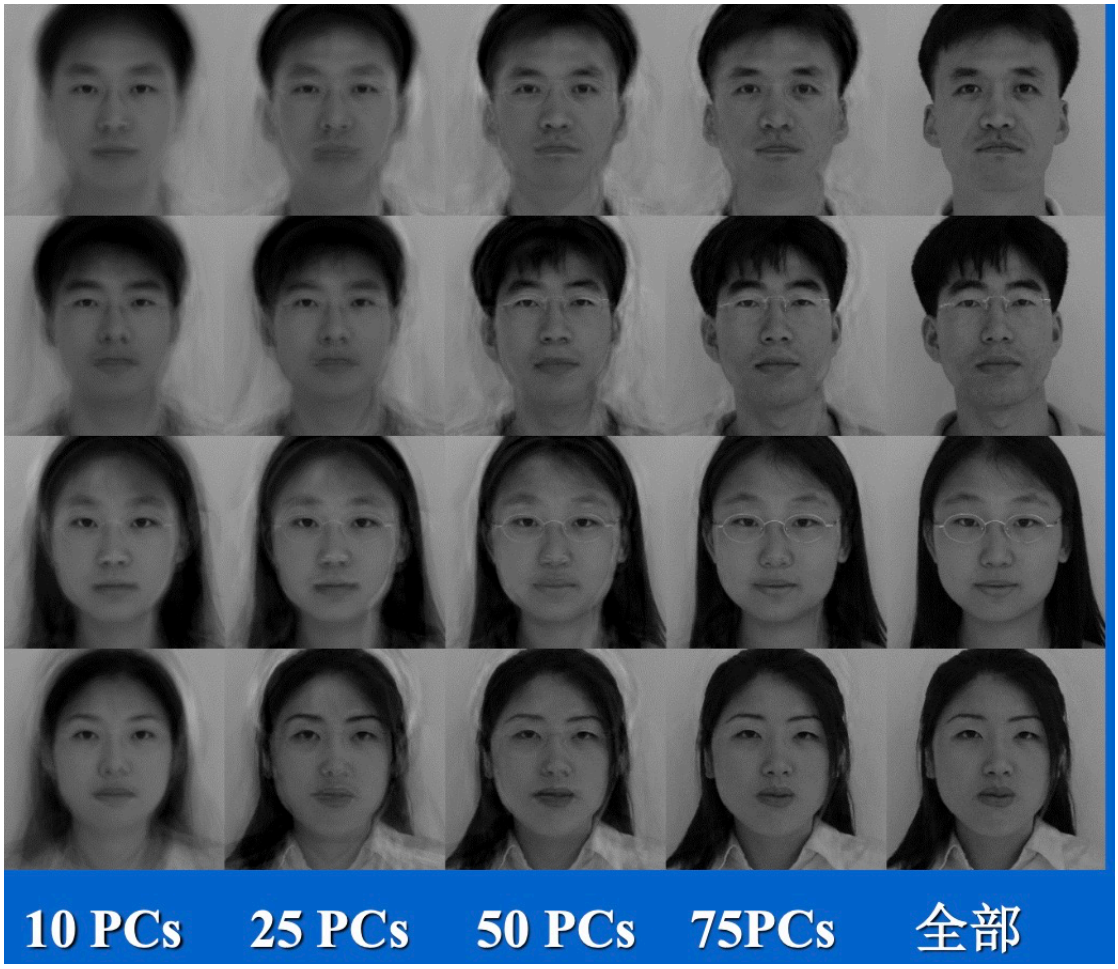
25个PCs重构的结果

50个PCs重构的结果

100个PCs重构的结果

美中不足的是，自己在PC较少的情况下重构出的人脸不够清晰，目前还没有找到具体原因，**可能是由于为了显示图像 的时候数据类型变化导致的。也可能是由于图像大小不够大，因此出现失真导致的**；**也可能是由于缩放的时候，图像比例没有找对，图像的形变或许对最终的结果也有一些影响**。

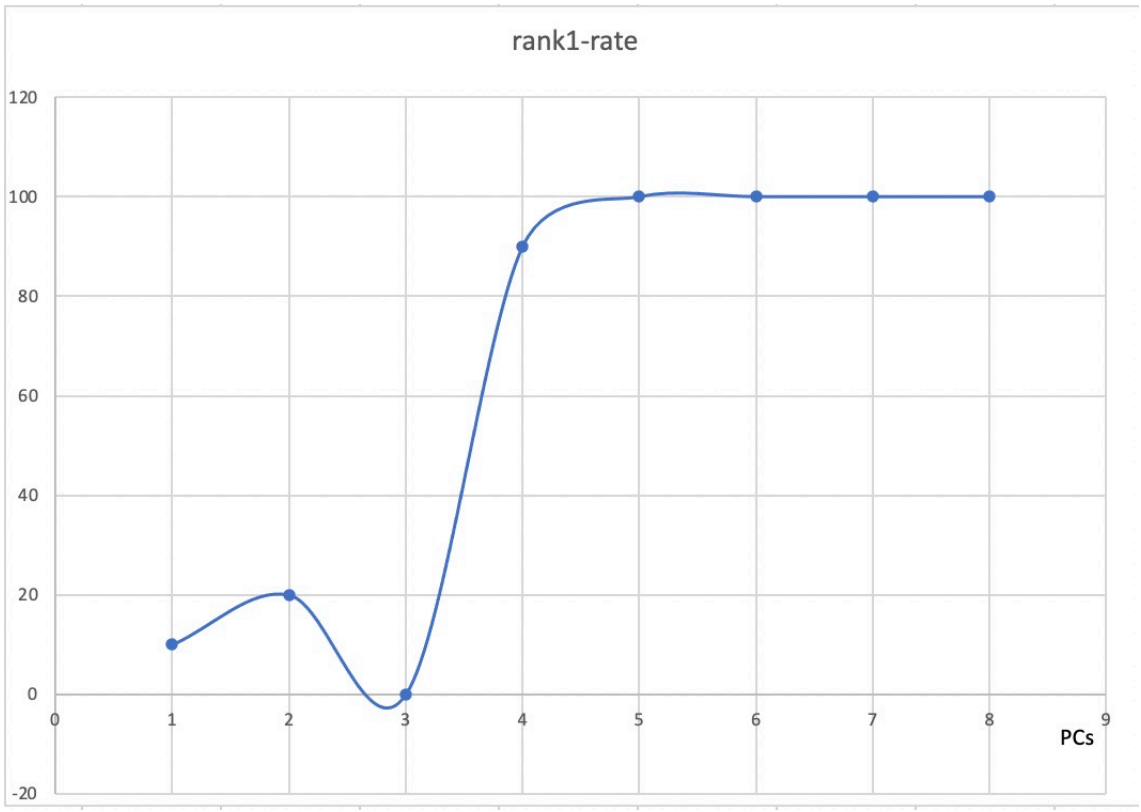
对比下图老师课堂上给的结果。自己实现的不同PC下的恢复图虽然不够清晰，**但仍然比较好的显示出了“眼镜”的变化**，类似于下图第三个人，我的人脸的的**眼镜也是从无到逐渐清晰**。实验中我也自己实现了快速的矩阵特征值运算，以期望保持原图，效果有所提升，不过这里没有展示。



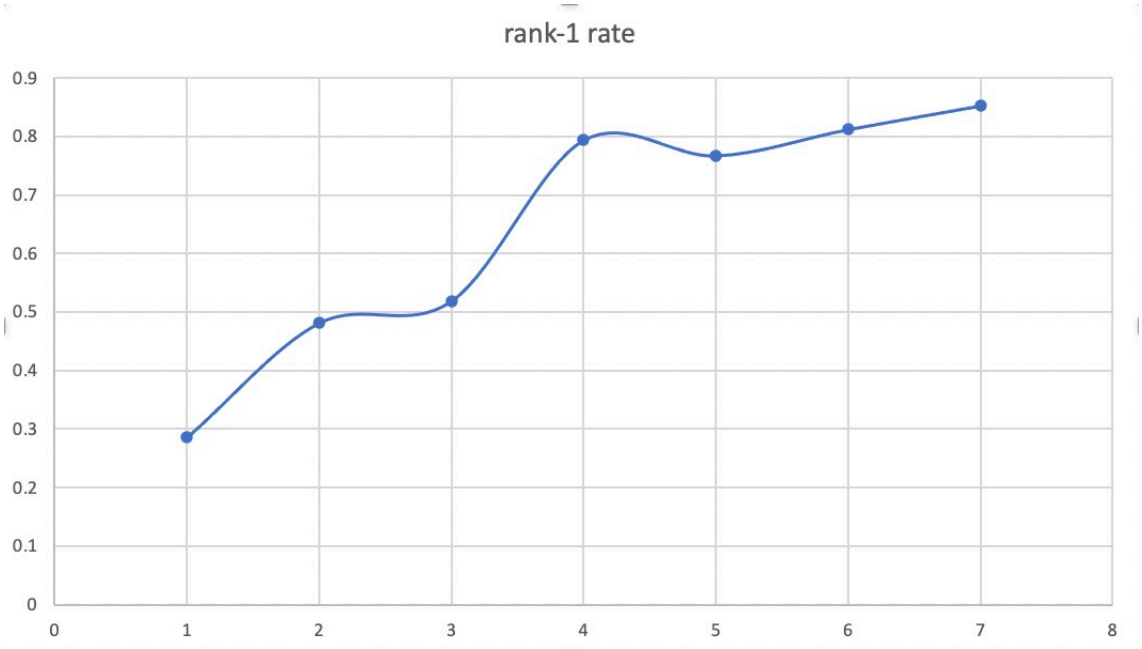
c. AT&T人脸库每 人的一半数据做训练，另一半做测试，展示随着PC增加，Rank-1识别率的变化曲线(即横坐标为使用的PC的数量、纵坐标为Rank-1 rate的一条曲线)

这里首先我用自己的人脸进行了验证，其中5张图片训练，9张图片(包括前面5张)作为匹配空间，20张额外的图片作为测试。经过手动的验证得到了以下结果

PCs	人物识别正确	rank1-rate
1	1	10
2	2	20
3	0	0
4	18	90
5	20	100
6	20	100
7	20	100
8	20	100

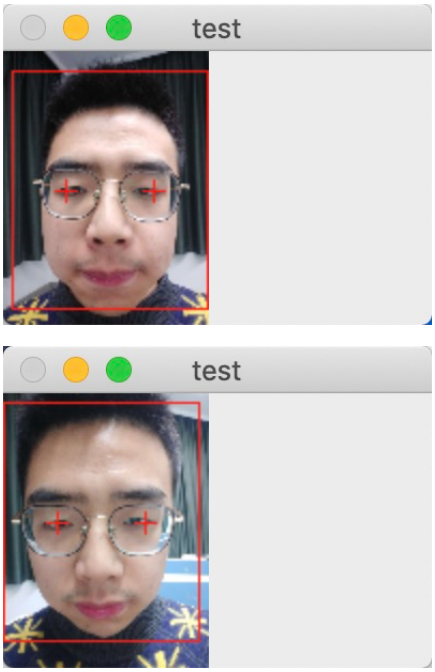


与此同时-我也用了AT&T的库进行测试，其中5张图片训练，5张图片测试。匹配空间即为训练的5张图片。得到如下数据.



后面的内容由于训练过于花费时间。我没有继续做，但对于100%的PC的情况，我得到的rank-1 rate为97.4325%

d. 个人图像的裁剪预处理过程



三、算法步骤描述

总体描述

- 获得人脸图像的训练集，通常为整个人脸数据 库;
- 对所有人脸图像作归一化处理;
- 通过PCA计算计算获得一组特征向量，通常一百个特征向量就足够
- 将每幅人脸图像都投影到由该组特征脸张成的子空间中，得到在该子空间坐标
- 对输入的一幅待测图像，归一化后，将其映射 到特征脸子空间中。然后用某种距离度量来描 述两幅人脸图像的相似性，如欧氏距离。

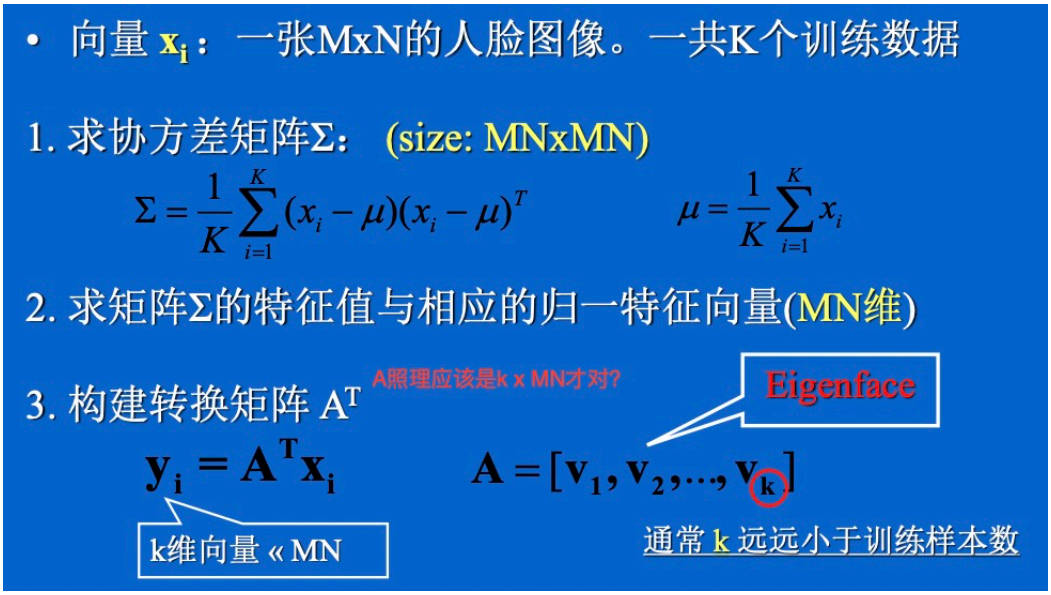
1. 图像预处理

首先对AT&T人脸库进行眼睛位置标注预处理。同时拍摄自己人脸图像，并对自己的人脸图像进行预处理。尽可能保证眼睛的在同一水平线上。

- 裁减个人图像中的键盘事件
  - '-' 缩小图片
  - '+' 放大图片
  - 'q'显示出crop的ROI，再按空格保存图片。
  - 'w', 's', 'a', 'd' 移动矩形模版
- 保存完预处理图片后，图像名称为{i}_roi.jpg
- 修改预处理代码，注释main函数的部分，调用toGray()
- 此时图像转为灰度图，并且名称为{i}.pgm，每个图像大小为92 x 112

注：由于预处理部分主要是手工操作，因此代码实现交互性不强。最终提交的二进制文件中没有包含其中，但源程序给出。

2. 训练过程



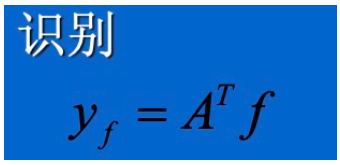
- 读入人脸图片，缩放，归一，加载数据集
- 计算均值矩阵和协方差矩阵-调用calcCovarMatrix()

- 计算协方差矩阵的特征值和特征向量-调用eigen
- 特征向量转换为特征脸，并将前十个特征值最大的特征脸拼接输出。
- 根据特征向量矩阵，选择一定数量的PCs，构建A_T
- 将模型相应参数保存，主要是energyPercent，图像文件名以及A_T矩阵的内容

3. 识别过程

下面讲述的为识别的过程算法，不包括人工的判断和计算以及一些辅助代码。

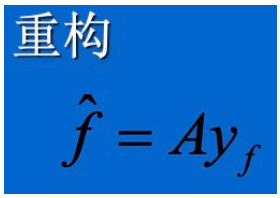
- 读入模型，读入人脸库数据
- 读入测试人脸，并将其映射到空间A_T；



- 将人脸库中人脸映射到空间A并缓存。
- 遍历测试映射后的测试人脸与人脸库中的人脸的欧式距离，求得最近的值
- 将二者混合。将结果输出。

4. 重构过程

- 读入模型
- 读入需要重构的人脸，将其映射到空间A_T
- 重新将映射后的结果反推，得到重构的图像向量



- 将向量重新转变为矩阵形式，显示输出。

四、算法实现要点(具体实现与程序描述)

此处讲解本人对算法的具体实现(包含注释与解析)，基本步骤与 三、算法步骤描述 中一致。

注意：程序运行的命令行参数在代码中给出，这里就不表示了。

1. 图像的预处理

- 显示红色模版

```
int ROIX = 0;
int ROIY = 0;
Point lefteye;
Point righteye;
```

```
Rect ROI;
const int ROIWIDTH = 92;
const int ROIHEIGHT = 112;
void showImageWithMask(Mat img, string windowName )
{
    setROI(img);
    rectangle(img, ROI, Scalar(0,0,255));
    line(img, lefteye - Point(5, 0), lefteye + Point(5, 0), Scalar(0, 0, 255));
    line(img, lefteye - Point(0, 5), lefteye + Point(0, 5), Scalar(0, 0, 255));
    line(img, righteye - Point(5, 0), righteye + Point(5, 0), Scalar(0, 0,
255));
    line(img, righteye - Point(0, 5), righteye + Point(0, 5), Scalar(0, 0,
255));
    namedWindow(windowName);
    imshow(windowName, img);
    //waitKey();
}
```

在其他步骤中确定ROI(即矩形的大小)以及眼睛的中心点，而后在图像中绘制矩形，同时绘制十字直线，并在图像上显示。此函数被 processImg()函数调用。

- 处理具体的图像

```
void processImg(Mat img)
{
    Mat i;
    img.copyTo(i);
    showImageWithMask(i);
    Mat img2;
    int t = waitKey(-1);
    switch (t) {
            //save:q
        case 113:
            cout << "save roi";
            saveROI(i);
            break;
            //+
        case 61:
            resize(img, img2, Size(img.cols * 1.1, img.rows * 1.1), 0, 0,
INTER_LINEAR);
            ROIX = img2.cols / 2 - ROIWIDTH / 2;
            ROIY = img2.rows / 2 - ROIHEIGHT / 2;
            destroyWindow("test");
            processImg(img2);
            break;

            //-
        case 45:
```

```
            resize(img, img2, Size(img.cols * 0.9, img.rows * 0.9), 0, 0,
INTER_LINEAR);
            ROIX = img2.cols / 2 - ROIWIDTH / 2;
            ROIY = img2.rows / 2 - ROIHEIGHT / 2;
            destroyWindow("test");
            processImg(img2);
            break;

            //key right
        case 100:
            ROIX++;
            destroyWindow("test");
            processImg(img);
            break;
            //key left
        case 97:
            ROIX--;
            destroyWindow("test");
            processImg(img);
            break;
            //key down
        case 115:
            ROIY--;
            destroyWindow("test");
            processImg(img);
            break;
            //key up
        case 119:
            ROIY++;
            destroyWindow("test");
            processImg(img);
            break;
        default :
            cout << "no matching key input";
            destroyWindow("test");
            processImg(img);
            break;

    }
}
```

此为图像处理的具体函数，即根据键盘输入，移动mask，而后crop出对应大小的图片。

- toGary函数

```
void toGray()
{
    Mat my_pic;
    for (int i = 1; i <= NUMPERPERSON; i++)
    {
        string path =
"/Users/evan/Desktop/Junior_fall_winter/CV/att_faces/s42/" + to_string(i) +
"_roi.jpg";
        my_pic = imread(path);
        Mat res;
        cvtColor(my_pic, res, COLOR_BGR2GRAY);
        imwrite("/Users/evan/Desktop/Junior_fall_winter/CV/att_faces/s42/" +
to_string(i) + ".pgm",res);
    }
}
```

toGray函数从对应的路径读入jpg图片，将其转化为灰度图，而后以pgm的形式输出。

2. 训练过程

- 加载数据集

```
//加载数据集
mytrain::mytrain(string datasetDir, int trainNum)
{
    if(trainNum > IMGPERPERSON)
    {
        cerr << "train number per person should no more than image per person";
    }
    string fileName;
    //遍历数据集的所有文件
    //添加对应的文件名
    //文件名格式为：si/j.pgm
    //最终路径为：datasetDir/si/j.pgm
    for (int i = 1; i <= PEOPLENUM; i++)
    {
        for(int j = 1; j <= trainNum; j++)
        {
            fileName = 's' + to_string(i) + "/" + to_string(j) + ".pgm";
            fileNames.push_back(fileName);
            Mat gray, imgResized;   //缩放用于加快特征值计算
            cvtColor(imread(datasetDir + fileName), gray, COLOR_BGR2GRAY);
            resize(gray, imgResized, Size(WIDTHRESIZED, HEIGHTRESIZED));
            dataset.push_back(imgResized);
        }
    }
}
```

根据命令行参数输入的路径，在对应位置读入图片，resize，并加载数据集。//这里的格式已经确定，因此格式化名字，并且读入内容即可。

- 训练--即计算A_T，输出特征脸，平均脸

```
void mytrain::train(double energyPerc)
{
    energyPercent = energyPerc;
    Mat covMat, mean;   //协方差矩阵 均值向量
    Mat eigenValues, eigenVectors;
    //计算协方差矩阵
    calcCovarMatrix(dataset, covMat, mean, COVAR_NORMAL);
    //covMat size: WIDTHRESIZED * HEIGHTRESIZED
    //根据协方差矩阵 计算得到排序的特征值与特征向量
    //特征向量s作为矩阵的形式
    eigen(covMat, eigenValues, eigenVectors);
    //根据能量百分比获得 前面一些特征值对应的特征向量
    A_T = eigenVectors.rowRange(0, (int)(eigenVectors.rows * energyPercent *
0.01));
    //A_T size:
    //WIDTHRESIZED * HEIGHTRESIZED * energeyPercent * 0.01 x WIDTHRESIZED *
HEIGHTRESIZED
    //检验结果的正确性
    cout << "A_T column is:" << A_T.cols << endl;
    cout << "A_T row is: " << A_T.rows << endl;
    //输出前10个特征脸
    std::vector<cv::Mat> firstTen;  //前十个特征脸
    cv::Mat firstTenConcat; //前十个特征脸连接
    for (int i = 0; i < 10; ++i)
    {
        cv::Mat tmp(cv::Size(WIDTHRESIZED, HEIGHTRESIZED), CV_64F);
        cv::Mat tmp_int(cv::Size(WIDTHRESIZED, HEIGHTRESIZED), CV_8UC1);
        for (int j = 0; j < WIDTHRESIZED * HEIGHTRESIZED; ++j)
        {
            tmp.at<double>(j / WIDTHRESIZED, j % WIDTHRESIZED) =
eigenVectors.at<double>(i, j);
        }
        cv::normalize(tmp, tmp, 255, 0, cv::NORM_MINMAX);
        tmp.convertTo(tmp_int, CV_8UC1);
        firstTen.push_back(tmp_int);
    }
    cv::hconcat(firstTen, firstTenConcat);
    cv::imwrite("firstTen.jpg", firstTenConcat);
    //计算平均脸并输出
    Mat aveFloat = Mat::zeros(HEIGHTRESIZED, WIDTHRESIZED, CV_64F);
    Mat ave_res = Mat::zeros(HEIGHTRESIZED, WIDTHRESIZED, CV_8UC1);

    int total = 10;        //需要平均的脸的总数
    vector<Mat> allFaces;  //所有的脸 浮点
```

```
    for (int i = 0; i < total; i++)
    {
        Mat tmp(Size(WIDTHRESIZED, HEIGHTRESIZED),CV_64F);
        //特征向量转换为特征脸
        for (int j = 0; j < WIDTHRESIZED * HEIGHTRESIZED; j++)
        {
            tmp.at<double>(j / WIDTHRESIZED, j%WIDTHRESIZED) =
eigenVectors.at<double>(i, j);
        }
        //normalize(tmp, tmp, 255, 0, NORM_MINMAX);
        allFaces.push_back(tmp);
    }
    for (int i = 0; i < total; i++)
    {
        aveFloat = aveFloat + allFaces.at(i);
    }
    aveFloat = aveFloat / (double)total;
    normalize(aveFloat, aveFloat, 255, 0, NORM_MINMAX);
    aveFloat.convertTo(ave_res, CV_8UC1);
    imwrite("ave_res.jpg",ave_res); //输出特征脸
}
```

通过调用calcCovarMatrix(dataset, covMat, mean, COVAR_NORMAL) 完成了协方差矩阵的计算。通过调用eigen(covMat, eigenValues, eigenVectors) 完成了特征值与特征向量的计算。选择一部分特征向量，形成A_T矩阵，即新的空间。最后特征脸转换输出。并遍历求和，平均，计算平均脸。

训练过程的参数模型输出这里就不表示了，主要就是energyPercent、**A_T**等。

3. 测试过程

测试过程有两种模式，一种为传入测试图片集合的路径，一种为直接传入测试图片。前一种方法用于手工计算rank-1 rate。后面也有所调整，用于自动化的判断，和计算AT&T集合上的rank-1 rate。下面讲解内容中不包含这部分内容，主要给出测试过程的算法部分。

- 数据集加载

```
mytest::mytest(string modelPath, string datasetDirectory)
{
    datasetDir = datasetDirectory;
    ifstream in(modelPath);
    double energyPercent;
    int fileSize;
    //从模型读入基本参数
    in >> WIDTHRESIZED >> HEIGHTRESIZED >> energyPercent >> fileSize;
    A_T = Mat(Size(WIDTHRESIZED * HEIGHTRESIZED, WIDTHRESIZED * HEIGHTRESIZED *
energyPercent*0.01), CV_64F);
    //A_T size: WIDTHRESIZED * HEIGHTRESIZED * energyPercent*0.01 x
WIDTHRESIZED * HEIGHTRESIZED
    for (int i = 0; i < fileSize; i++)
```

```
    {
        string fileName;
        in >> fileName;
        fileNames.push_back(fileName);
    }
    //读入A_T矩阵
    for (int i = 0; i < A_T.rows; i++)
    {
        for (int j = 0; j < A_T.cols; j++)
        {
            in >> A_T.at<double>(i, j);
        }
    }

    //遍历数据集的所有文件
    //添加对应的文件名
    //文件名格式为：si/j.pgm
    //最终路径为：datasetDir/si/j.pgm
    string fileName;
    fileNames.clear();
    for (int i = 1; i <= PEOPLENUM; i++)
    {
        for(int j = 1; j <= IMGPERPERSON - 1; j++) //1-到9张图片-第十章不算
        {
            fileName = 's' + to_string(i) + "/" + to_string(j) + ".pgm";
            fileNames.push_back(fileName);
            Mat img, imgResized;   //缩放用于加快特征值计算
            cvtColor(imread(datasetDir + fileName), img, COLOR_BGR2GRAY);
            resize(img, imgResized, Size(WIDTHRESIZED, HEIGHTRESIZED));
            imgResized.reshape(0, WIDTHRESIZED *
HEIGHTRESIZED).convertTo(imgResized, CV_64F);
            Mat Y_T = A_T * imgResized; //同样转换为新的基下的表示
            dataset.push_back(Y_T);
        }
    }
}
```

对于每个人-前面5张训练，第1张到第9张作为识别(test)到时候查找的范围--即识别匹配的时候-仅仅从1-到9张选择。//这是针对单一输入图片的人工验证。

数据集加载过程比较容易，主要就是model的读入以及数据集图片的读入。最为重要的是，图片的映射-——将其映射到新的基下。

针对rank-1图绘制的测试函数。我给出代码mytestRank，基本结果与mytest相同，但有一些区别！同时这个也是手工调整的！

- 测试实现

```
//读入测试图片
//识别-找到最相似图片-输出
//同时将识别结果叠加在输入的人脸图像上
void mytest::test(string filePath)
{
    Mat imgTestOri, imgTest;
    imgTestOri = imread(filePath);
    cvtColor(imgTestOri, imgTest, COLOR_BGR2GRAY);
    resize(imgTest, imgTest, Size(WIDTHRESIZED, HEIGHTRESIZED));
    //转变为向量
    imgTest.reshape(0, WIDTHRESIZED * HEIGHTRESIZED).convertTo(imgTest,
CV_64F);
    Mat mapped = A_T * imgTest;
    double min = -1;
    int index = 0;
    //计算欧式距离-找到最相近的图片
    for (int i = 0; i < dataset.size(); i++)
    {
        Mat diff = mapped - dataset[i];
        double mo = norm(diff);
        if(mo < min || min < 0)
        {
            min = mo;
            index = i;
        }
    }
    //输出相关结果
    cout << min;
    cout << "index is : " << index << endl;
    cout << "file name is : " << fileNames[index];
    Mat find = imread(datasetDir + fileNames[index]); //仅仅在前面用于制作特征向量的
里面找-不合理-应该在所有的数据集里面找
    Mat blend;
    addWeighted(imgTestOri, 0.5, find, 0.4, 0, blend);
    imwrite("test_" + to_string(countnum) +".jpg", imgTestOri);
    imwrite("blend_" + to_string(countnum) + ".jpg", blend);
    imwrite("findMatch_" + to_string(countnum) + ".jpg", find);
    countnum++;
    namedWindow("blend Image");
    imshow("blend Image", blend);
    waitKey();
}
```

对于读入的图片，将其映射到特征基上，随后遍历数据集，计算欧式距离，其中欧式距离最短的即为我们需要的内容。最后将得到的图片混合，并输出对应结果。