浙江大学

本科实验报告

Eigenface 人脸识别算法

课程名称：	计算机视觉
姓名：	夏豪诚
学院：	计算机科学与技术学院
专业：	信息安全
学号：	**3170102492**
指导老师：	宋明黎

2019 年 12 月 14 日

浙江大学实验报告

专业：信息安全
姓名：夏豪诚
学号：3170102492
日期：2019 年 12 月 14 日
地点：无

课程名称：计算机视觉　指导老师：宋明黎　成绩：
实验名称：Eigenface 人脸识别算法　实验类型：综合实验　同组学生姓名：无

一、 实验环境

表 1: 测试环境

item	detail
CPU	Intel® Core™ i7-6700HQ CPU 2.60GHz
RAM	16.0GB DDR4 2133MHz
hard disk	SSD 256GB
OS	Windows 10 Pro 64-bit
IDE	Microsoft Visual Studio Community 2019
OpenCV	3.4.5

二、 实验目的和要求

1. 实验目的

熟悉 Eigenface 人脸识别算法。

2. 基本要求

Eigenface 人脸识别算法自己写代码实现 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 或其他的；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。

三、 实验内容和步骤

1. 实验内容

(1) 实现两个类，Face 类来进行对单张人脸进行操作；FaceLModel 类来进行 model 的读入和对人脸的统一操作。

(2) mytrain.cpp 实现了协方差矩阵和特征矩阵的计算，生成特征脸，并将前 10 个特征脸拼成一张图像，然后显示出来。

(3) mytest.cpp 实现了用特征矩阵对图片进行降维，然后计算欧氏距离进行人脸类型的判断。

实验中使用了 GitHub 上的人脸库 EigenFace ，共 40 个人，每人 10 张人脸。根据实验要求添加了自己的人脸图片 (保存于“42”文件夹中)，并额外添加了另一名同学的人脸图，最终使用的人脸库共 42 个人，每人 10 张人脸，对应于每张图片使用一个与图像文件名相同但后缀名为 txt 的文本文件 (文件内容为用空格隔开的四个整数) 标注每张图像的眼睛位置。

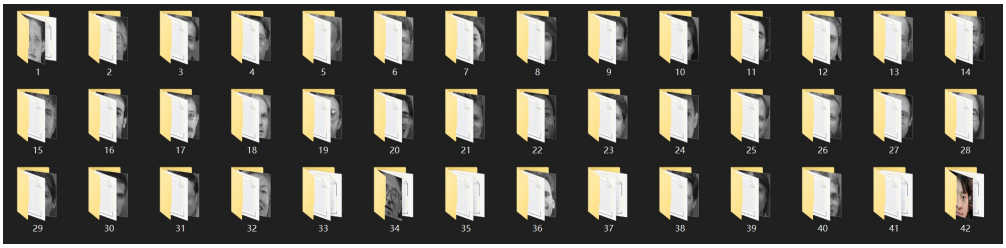


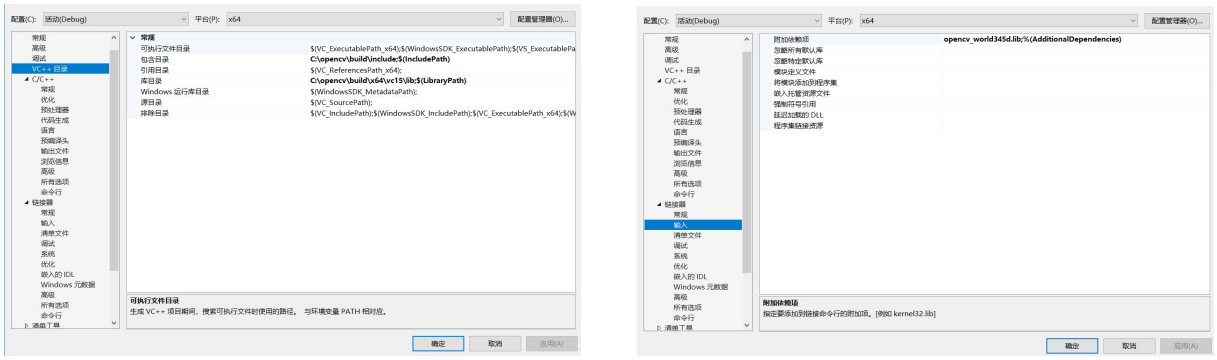
图 1: 数据集概览

针对这三个部分的具体代码实现将在实验步骤中进行详细说明。

2. 实验步骤

3.2.0 实验环境配置

环境配置与之前实验相同，在 Windows 10 操作系统的环境下，我们可以在官网下载 exe 后缀的可执行文件，运行后可以将编译好的对应版本 opencv 解压到制定的目录下，在本次实验中我选择的版本为 3.4.5，解压完成后为了可以在 VS 中调用 opencv 库函数我们需要修改 VC++ 目录下的包含目录和库目录添加 opencv 的解压位置，而后在链接器-> 输入-> 附加依赖项中添加 lib 文件。



(a) 配置图 1.　　(b) 配置图 2.

图 2: VS 配置界面

3.2.1 类的实现

头文件 FaceOp.h 中的类定义如下：

```
// File: FaceOp.h
#include <opencv2/opencv.hpp>
#include <iostream>
#include <string>
#include <vector>

#define MYEXT ".png"

const int WIDTH = 73;
const int HEIGHT = 89;
class Face {
public:
  cv::Mat originFaceImg;
  cv::Mat grayFaceImg;
  cv::Mat transformedFaceImg;
  int x1, y1, x2, y2;
  cv::Mat trans_mat;
  cv::Mat_<double> equalized_mat;
  cv::Mat_<double> vect;
  void load(std::string& path, std::string ext); //
      载入图像，得到originFaceImg和grayFaceImg，并进行transform操作
  void load(std::string& path); // load重载函数：可以在载入时将后缀附在path中
  void load_eye_pos(std::string& path); // 载入眼睛位置信息
  void transform(); // 眼睛位置作为中心位置进行旋转
};

class FaceModel {
public:
  int num_of_faces = 210;
  int num_of_persons = 42;
  int faces_per_person = 5;
```

```
    std::vector<Face*> faces;
    std::vector<cv::Mat_<double>> _samples;
    cv::Mat_<double> samples;
    void load(std::string& path);//读入模型
};
```

具体类中函数和方法的实现见源代码。

3.2.2 mytrain

训练过程简单分为四个步骤：

(1) 读入人脸图像后，计算协方差矩阵；

此处固定数据集的路径为绝对路径D://data。

```
const char* model_name = (char*)"default";
  double energy = 0.95;
  if (argc <= 2) {
    std::cout << "usage: mytrain.exe [energyPercent] [modelFileName] <faceSetDirPath>
        " << std::endl;
    std::cout << "<> representing optional para" << endl;
    return 0;
  }
  else if (argc >= 3) {
    model_name = argv[2];
    energy = atof(argv[1]);
  }
  std::string dirPath("D://data");
  if (argc >= 4) {
    dirPath = argv[3];
  }
  FaceModel facemodel;

  facemodel.load(dirPath);
  Mat samples, cov_mat, mean_mat;
  facemodel.samples.copyTo(samples);
  //cout << samples << endl;
  //cout << samples.size() << endl;
  cout << "Step[1/4]: Calculating Covariance Mat..." << endl;
  calcCovarMatrix(samples, cov_mat, mean_mat, CV_COVAR_ROWS | CV_COVAR_NORMAL);
  cout << mean_mat.size() << endl;
  //cout << cov_mat << endl;
  //cout << cov_mat.size() << endl;
  cov_mat = cov_mat / (samples.rows - 1);
```

(2) 计算特征矩阵，并通过能量百分比来计算出应该取多少特征脸；

```
cout << "Step[2/4]: Calculating Eigen Vector..." << endl;
  eigen(cov_mat, e_value_mat, e_vector_mat);
```

```
//cout << "eigen size " << e_value_mat.size() << endl;
//cout << e_value_mat << endl;
for (int i = 0; i < samples.rows; ++i) {
  samples.row(i) -= mean_mat;
}

double value_sum = sum(e_value_mat)[0];
//cout << e_vector_mat.size() << endl;
double energy_level = value_sum * energy;
double energy_sum = 0;
int k = 0;
//cout << "Log: <before energy_sum> e_value_mat.rows = " << e_value_mat.rows <<
    endl;
for (k = 0; k < e_value_mat.rows; k++)
{
  energy_sum += e_value_mat.at<double>(k, 0);
  if (energy_sum >= energy_level) break;
}
cout << k << endl;
e_vector_mat = (samples * e_vector_mat.t()).t();
e_vector_mat = e_vector_mat.rowRange(0, k);
e_value_mat = e_value_mat.rowRange(0, k);
```

(3) 运算的结果导出，保存为 model；

```
cout << "Step[3/4]: Save model..." << endl;
  FileStorage model(model_name, FileStorage::WRITE);
  model << "e_vector_mat" << e_vector_mat;
  model << "e_value_mat" << e_value_mat;
  model.release();
```

需要注意的是在此处的 FileStorage 对象创建时使用的 flag 设为FileStorage::WRITE |FileStorage::BASE64，这样我们保存的数据为二进制，可以减少存储压力，

图 3: model 文件内容示意

(4) 选出特征值最大的 10 张特征脸合并到一张图片并显示出来；

```
cout << "Step[3/4]:Output Top Ten EigenFace..." << endl;
//cout<<"Log: <before toImg> e_value_mat.rows = "<< e_value_mat.rows<<endl;
vector<Mat> Top10EigenFace;
for (int i = 0; i < 10; ++i) {
  Top10EigenFace.push_back(toImg(e_vector_mat.row(i), WIDTH, HEIGHT));
}
Mat result;
hconcat(Top10EigenFace, result);
result.convertTo(result, CV_8U, 255);
imshow("Top Ten EigenFace", result);
imwrite("Top Ten EigenFace.png", result);
```

3.2.3 mytest

识别过程简单分为三个步骤：

(1) 将所有人脸库中的训练脸都转换成特征脸为基的低维向量;

```
if (argc <= 2) {
  cout << "usage: mytest.exe [testFaceImagePath] [modelPath] <faceSetDirPath> " <<
      endl;
  cout << "Tips: <> representing optional para" << endl;
  return 0;
}
```

```
FaceModel facemodel;
std::string dirPath("D://data");
facemodel.load(dirPath);

char* model_name = (char*)"default";
char* file_name = (char*)"default";

if (argc >= 3) {
  model_name = argv[2];
  file_name = argv[1];
}
if (argc >= 4) {
  // has dirPath
  dirPath = argv[3];
}

cout << "Step[1/3]: Get low dimension vector..." << endl;
FileStorage model(model_name, FileStorage::READ | FileStorage::BASE64);
Mat e_vector_mat, e_value_mat;
model["e_vector_mat"] >> e_vector_mat;
model["e_value_mat"] >> e_value_mat;
Mat distance;
Mat samples;
Face face;
facemodel.samples.copyTo(samples);
distance = e_vector_mat * samples;
```

(2) 计算和训练库中各张人脸的 NORM_L2(即欧氏距离)，找出最小的距离的对应图片，将其作为训练集中我们识别得到的最为接近待识别图的图片；

```
for (int _i = 1; _i <= facemodel.num_of_persons; ++_i) {
  for (int _j = 1; _j <= facemodel.faces_per_person_total; _j++)
  {
    string facePath("D://data/" + to_string(_i) + "/" + (_j < 10 ? '0' +
        std::to_string(_j) : std::to_string(_j)));
    cout << "Log: facePath = " << facePath << endl;

    face.load(facePath, ".png");
    Mat face_vect = e_vector_mat * face.vect;
    double min_d = norm(face_vect, distance.col(0), NORM_L2);
    double temp_d = 0;
    int min_i = 0;
    cout << "Log: distance.cols" << distance.cols <<endl;
    for (int i = 1; i < distance.cols; ++i) {
      temp_d = norm(face_vect, distance.col(i), NORM_L2);

      if (temp_d <= min_d) {
        min_d = temp_d;
        min_i = i;
```

```
      }
    }
    cout << (min_i / facemodel.faces_per_person) + 1 << "/" << (min_i %
        facemodel.faces_per_person) + 1 << " ";
  }
  cout << "Log: faceLoad complete" << endl;
  cout << endl;
}
string fileNameStr(file_name);
face.load(fileNameStr);
cout << "Log: fileNameStr = " << fileNameStr << endl;
cout << "Log: test pic loaded" << endl;
Mat face_vect = e_vector_mat * face.vect;
double min_d = norm(face_vect, distance.col(0), NORM_L2);
double temp_d = 0;
int min_i = 0;

for (int i = 1; i < distance.cols; ++i) {
  temp_d = norm(face_vect, distance.col(i), NORM_L2);
  cout << "i = " << i << "distance = " << temp_d << endl;
  if (temp_d <= min_d) {
    min_d = temp_d;
    min_i = i;
  }
}
cout << "min_i->" << min_i << endl;
cout << (min_i / facemodel.faces_per_person) + 1 << "/" << (min_i %
    facemodel.faces_per_person + 1) << " " << endl;
```

(3) 然后打上标记，显示图片。

```
Mat origin_mat = face.originFaceImg;
  Mat similar_mat = facemodel.faces.at(min_i)->originFaceImg;
  string text = "s" + to_string(min_i / facemodel.faces_per_person + 1) + " No." +
      to_string(min_i % facemodel.faces_per_person + 1);
  cout << text << endl;
  putText(origin_mat, text, Point(10, 20), CV_FONT_HERSHEY_SIMPLEX, 0.5, (0, 0, 255),
      2, 8);
  imshow("OriginFaceWithMarker", origin_mat);
  imshow("Similar Pic", similar_mat);
```

四、 主要仪器设备

计算机，Visual Studio 2019

五、 实验结果

1. 编译运行

在 VS2019 中编译成功后，在控制台运行可执行程序 mytrain.exe 和 mytest.exe，并指定对应参数。mytrain.exe 设定能量百分比和 model 文件名后运行效果如图所示：

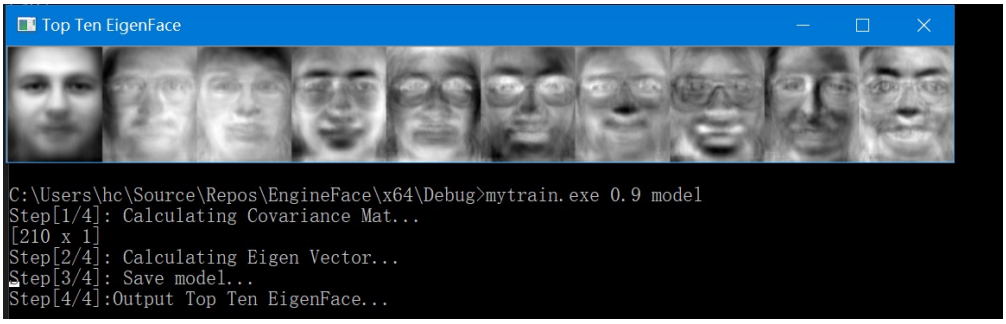


图 4: mytrain.exe 运行效果

mytest.exe 设定待识别文件路径和 model 路径后运行效果如图所示：

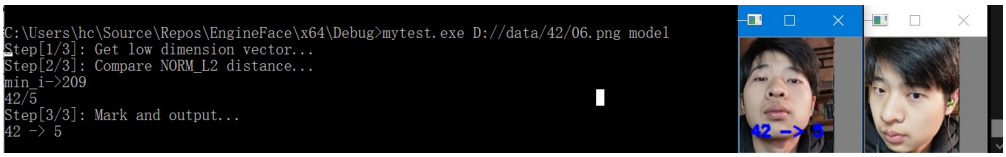


图 5: mytest.exe 运行效果

2. 文件变化

我们首先查看目录下的文件，看到 mytrain.exe 运行的结果，model 文件model和前 10 个特征脸的拼接图像Top Ten EigenFace.png。

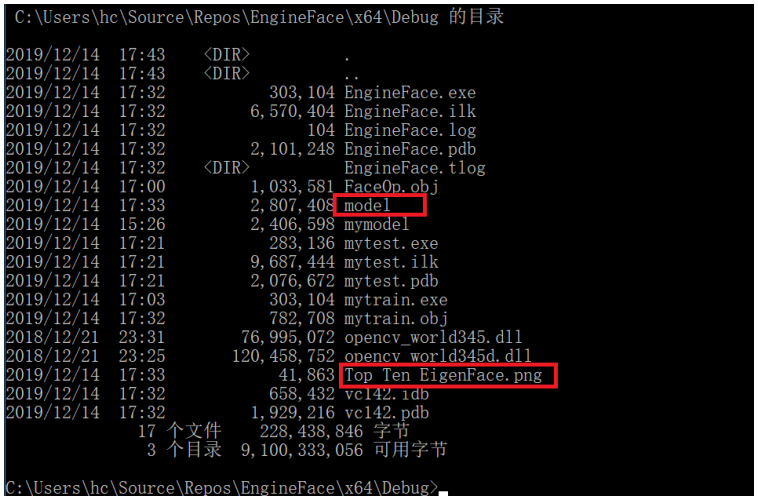


图 6: 文件变化

3. 运行结果

可以看到前 10 个特征脸的拼接图像Top Ten EigenFace.png如下：

图 7: 前 10 个特征脸拼接图像

如下是几组识别结果：

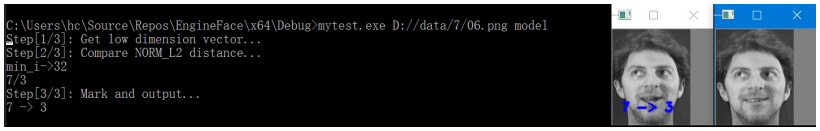


图 8: 识别结果 1

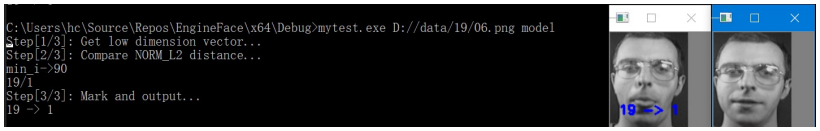


图 9: 识别结果 2

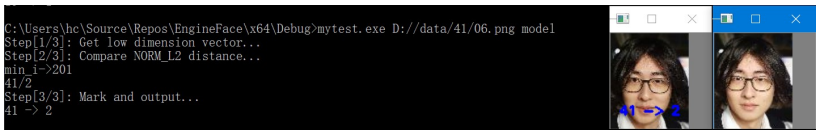


图 10: 识别结果 3

下方是对我的人脸图像的识别结果。

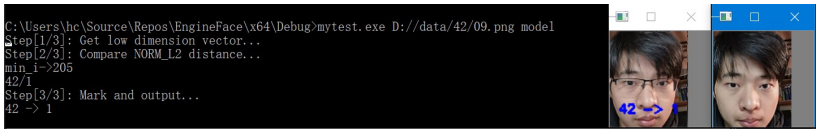


图 11: 识别结果 4

六、 实验结果分析

在本次实验中总体而言比前两次实验都要复杂一些，最后的结果相对较好，但是在进行批量测试的时候也出现了识别不正确的现象，本次实验数据集中的“14”文件夹和“27”文件夹下的人脸图像就是如此，从肉眼观察看，确实两者人像比较相似但还是可以轻易辨别，不过在识别中却出现了识别错误的结果。

本次实验与上一次实验一样激发了我对问题的探索意识和解决问题的意识，尽管最后没有做到针对数据集完美识别，但这个有趣的实验还是极大地激发了我对计算机视觉的好奇与兴趣，尝试添加了身边的同学的人脸数据来进行训练，希望能在之后的课程中学到更多的知识。