浙江大学

计算机视觉作业报告

作业名称：实现 EigenFace

姓　　名：陈润健

学　　号：3160103989

电子邮箱：3160103989@zju.edu.cn

联系电话：18868104871

导　　师：潘纲

2018 年　　12 月　　18　日

实现 EigenFace

一、 作业已实现的功能简述及运行简要说明

1. 实现 EigenFace。
2. 使用简单的画图工具标定眼睛位置。
3. 对图像进行简单的预处理：均衡化，简单的滤波等等。
4. 训练。
5. 识别，并进行实验，探求识别率与 PC 数的关系。
6. 重构。

二、 作业的开发与运行环境

IDE：Xcode 10.1

操作系统：MacOS 10.14.1

SDK：macosx10.14

三、 基本思路，用到的函数及流程

1. EigenFace 的基本思路：
   1) 训练：将输入的所有人脸取平均，对于每一个人脸，减掉平均脸之后拉成一列，拼成一个矩阵ϕ，然后构造协方差矩阵$\phi\phi^T$，求出协方差矩阵的特征向量，取前 n 个构造转换矩阵 A，然后将每一张人脸通过矩阵 A 投影到 n 维空间中，存入数据库。（由于协方差矩阵的维度太大，可以考虑采用 SVD 法求解特征值和特征向量）
   2) 识别：将输入的新的人脸通过 A 矩阵投影到 n 维空间中，找到数据库中在 n 维空间中与新的人脸最接近的一个即可。
   3) 重构：将输入的新的人脸减掉平均脸，然后投影到 n 维空间中，以这 n 个维度上的数值作为系数，乘上 n 张特征脸，再加上平均脸，就变成了重构出来的结果。

2. 程序的构架：

1) 类的设计：

```
class ChenRJ_EigenFace
{
private:
    int Subspace_Dimension,Common_H,Common_W;//dimension of subspace   common Height and Width for data

    Mat Transform_to_Subspace;//transform matrix A

    vector<Mat> Origin_Data;//original data

    vector<Mat> Database;//database

    Mat im_Avg;//average image

    void Pre_Process();//pre process

    void Train();//train

    void Calculate_Average(vector<float> *Average_Face);

    void Calculate_Trans_Matrix(int Subspace_Dimension,vector<float> *Average_Face);

    int Get_Index(int Height,int Width);

public:
    ChenRJ_EigenFace(vector<Mat> *Training_data,double Energy_Rate);
    ~ChenRJ_EigenFace(){};
    int Indentification(Mat New_face);//identification
    Mat Get_Face_from_Database(int Index);
    Mat Reconstruct(Mat Reconstruct_Source,double chosen_rate);//reconstruct
};
```

2) 构造函数：初始化原始数据库，数据库以及各个参数（子空间维度，照片的大小，转换矩阵 A），然后调用训练函数。

```
ChenRJ_EigenFace::ChenRJ_EigenFace(vector<Mat> *Training_data,double Energy_Rate)
{
    Origin_Data = *Training_data;
    Subspace_Dimension = Max_Featured_Face*Energy_Rate;
    Common_H = H;
    Common_W = W;
    Transform_to_Subspace = Mat(Subspace_Dimension,Common_H*Common_W,CV_32FC1);
    Database.resize(Origin_Data.size());

    for (int i = 0 ; i < Database.size() ; i++)
    {
        Database[i] = Mat(Subspace_Dimension,1,CV_32FC1);
    }
    Train();
}
```

3) 训练函数：

```
void ChenRJ_EigenFace::Train()
{
    Pre_Process();

    vector<float> *Average_Face = new vector<float>(Common_H*Common_W);

    Calculate_Average(Average_Face);//calculate average face

    Calculate_Trans_Matrix(Subspace_Dimension,Average_Face);//transform matrix calculator

}
```

4) 计算平均值函数：

```
39 void ChenRJ_EigenFace::Calculate_Average(vector<float> *Average_Face)
40 {
41     for (int j = 0 ; j < H ; j++)
42         for (int k = 0 ; k < W ; k++)
43         {
44             (*Average_Face)[Get_Index(j,k)] = 0 ;
45         }
46     for (int i = 0 ; i < Origin_Data.size() ; i++)
47     {
48         for (int j = 0 ; j < H ; j++)
49             for (int k = 0 ; k < W ; k++)
50             {
51                 (*Average_Face)[Get_Index(j,k)] += static_cast<float>(Origin_Data[i].ptr<unsigned
                       char>(j)[k*3])+static_cast<float>(Origin_Data[i].ptr<unsigned char>(j)[k*3+1])
                       +static_cast<float>(Origin_Data[i].ptr<unsigned char>(j)[k*3+2]);
52
53             }
54     }
55     for (int j = 0 ; j < H ; j++)
56         for (int k = 0 ; k < W ; k++)
57         {
58             (*Average_Face)[Get_Index(j,k)] = (*Average_Face)[Get_Index(j,k)]/Origin_Data.size();
59             (*Average_Face)[Get_Index(j,k)] = (*Average_Face)[Get_Index(j,k)]/3.0;
60         }
61 }
```

5) 计算转换矩阵：

```
164 void ChenRJ_EigenFace::Calculate_Trans_Matrix(int Dimension, vector<float> *Average_Face)
165 {
166     Mat Cov = Mat(Common_H*Common_W,Origin_Data.size(),CV_32FC1); ⚠ Implicit conversion loses integer precision: 'std::_1::vector<...
167     //matrix fine
168
169     im_Avg = Mat(H,W,CV_32FC1);
170     //average image
171
172     //get average image
173     for (int j = 0 ; j < Common_H ; j++)
174         for (int k = 0 ; k < Common_W ; k++)
175         {
176             float * pos = im_Avg.ptr<float>(j);
177             pos[k] =  static_cast<float>((*Average_Face)[j*Common_W+k]);
178         }
179
180     //calculate matrix fine
181     for (int i = 0 ; i < Origin_Data.size() ; i++)
182     {
183         for (int j = 0 ; j < Common_H ; j++)
184             for (int k = 0 ; k < Common_W ; k++)
185             {
186                 float * pos = im_Avg.ptr<float>(j);
187                 Cov.ptr<float>(j*Common_W+k)[i] = (static_cast<float>(Origin_Data[i].ptr<unsigned char>(j)[k*3])
                       +static_cast<float>(Origin_Data[i].ptr<unsigned char>(j)[k*3+1])
                       +static_cast<float>(Origin_Data[i].ptr<unsigned char>(j)[k*3+2]))/3 - pos[k];
188             }
189     }
190
191     //calculate eigen vectors and values
192
193     Mat C = Cov.t()*Cov;
```

```
    Mat eValuesMat;
    Mat eVectorsMat;

    eigen(C, eValuesMat, eVectorsMat);

    //select eigen vectors

    vector<Mat> Eigen_Vectors = *new vector<Mat>(Subspace_Dimension);

    for (int i = 0 ; i < Subspace_Dimension ; i++)
    {
        Eigen_Vectors[i] = Mat(Origin_Data.size(),1,CV_32FC1); ⚠ Implicit conversion loses integer precision: 'std::__1::vector<cv::...

        Eigen_Vectors[i] = eVectorsMat.row(i).t();

    }

    //calculate transform matrix

    for (int i = 0 ; i < Subspace_Dimension ; i++)
    {
        Mat now = Mat(Common_W*Common_H,1,CV_32FC1);
        now = (Cov*Eigen_Vectors[i]);
        normalize(now.t(),Transform_to_Subspace.row(i),1,0,NORM_L1);
    }

    //generate database
    for (int i = 0 ; i < Origin_Data.size() ; i++)
    {
        Mat now = Mat(Common_W*Common_H,1,CV_32FC1);
        for (int j = 0 ; j < Common_H ; j++)
            for (int k = 0 ; k < Common_W ; k++)

            {
                now.ptr<float>(j*Common_W+k)[0] = static_cast<float>(Origin_Data[i].ptr<unsigned char>(j)[k*3]);
            }
        Database[i] = Transform_to_Subspace*now;
    }

}
```

6） 识别函数：

```
int ChenRJ_EigenFace::Indentification(Mat New_Face)
{
    int Index = -1 ;
    double min = 100000000;

    Mat now = Mat(Common_W*Common_H,1,CV_32FC1);
    for (int j = 0 ; j < Common_H ; j++)
        for (int k = 0 ; k < Common_W ; k++)
        {
            now.ptr<float>(j*Common_W+k)[0] = static_cast<float>(New_Face.ptr<unsigned char>(j)[k*3])-
                im_Avg.ptr<float>(j)[k];;
        }

    Mat Data = Mat(Subspace_Dimension,1,CV_32FC1);
    Data = Transform_to_Subspace*now;

    for (int i = 0 ; i < Origin_Data.size() ; i++)
    {
        if (norm(Data,Database[i]) < min)
        {
            Index = i;
            min = norm(Data,Database[i]);
        }
    }
    return Index;
}
```

7） 重构函数：

```
Mat ChenRJ_EigenFace::Reconstruct(Mat Reconstruct_Source,double chosen_rate)
{
    Mat New_Face,temp2;

    Mat now = Mat(Common_W*Common_H,1,CV_32FC1);

    for (int j = 0 ; j < Common_H ; j++)
        for (int k = 0 ; k < Common_W ; k++)
        {
            now.ptr<float>(j*Common_W+k)[0] = static_cast<float>(Reconstruct_Source.ptr<unsigned char>(j)[k*3])-
                im_Avg.ptr<float>(j)[k];
        }

    Mat Data = Mat(Subspace_Dimension,1,CV_32FC1);

    Data = Transform_to_Subspace*now;

    int chosen_num = Subspace_Dimension*chosen_rate;

    Mat chosen_faces = Mat(chosen_num,1,CV_32FC1);

    for (int i = 0 ; i < chosen_num ; i++)
    {
        chosen_faces.ptr<float>(i)[0] = Data.ptr<float>(i)[0];
    }

    temp2 = Mat(Common_H,Common_W,CV_32FC1);
    for (int j = 0 ; j < Common_H ; j++)
        for (int k = 0 ; k < Common_W ; k++)
        {
            temp2.ptr<float>(j)[k] = 0;
        }
```

```
    for (int i = 0 ; i < chosen_num ; i++)
    {
        Mat row_n = Transform_to_Subspace.row(i),row_now;
        normalize(Transform_to_Subspace.row(i),row_now,0,255,NORM_MINMAX);

        for (int j = 0 ; j < Common_H ; j++)
            for (int k = 0 ; k < Common_W ; k++)
            {
                temp2.ptr<float>(j)[k] = temp2.ptr<float>(j)[k]+chosen_faces.ptr<float>(i)
                    [0]*((row_now).ptr<float>(0)[j*Common_W+k]);
            }
    }

    temp2 += im_Avg;

    normalize(temp2,New_Face,0,255,NORM_MINMAX);

    return New_Face;
}
```

8） 演示程序（在 main 中）：

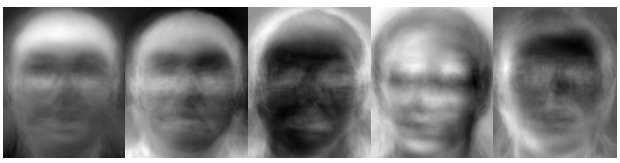
```
        //reconstruct test
        /*
        filename = "/Users/chenrj/Desktop/视觉/att_faces/s41/1.pgm";
        Mat Source = imread(filename);
        Mat re = new_database.Reconstruct(Source,1);
        imwrite("/Users/chenrj/Desktop/1.jpg",re);
        */


        //identification test
        /*
        filename = "/Users/chenrj/Desktop/视觉/att_faces/s41/1.pgm";
        Mat Source = imread(filename);
        int Index = new_database.Indentification(Source);

        string str = to_string((Index+1)/9+1);
        Point ID_pos(40,40);
        putText(Source, str, ID_pos,CV_FONT_HERSHEY_COMPLEX, 0.5, CvScalar(0,0,255));

        imwrite("/Users/chenrj/Desktop/1.jpg",Source);
        imwrite("/Users/chenrj/Desktop/2.jpg",new_database.Get_Face_from_Database(Index));
        */
```

四、 实验结果与分析：

1. 平均脸以及前 10 个特征脸（其他结果放在压缩文件中的实验结果文件夹中，麻烦老师检查）：

1） 平均脸：

2） 前十个特征脸：



3) 前十个特征脸相加（requirement 中的要求）：

2. 重构结果：

1) 自己的照片：第一张照片为没有经过训练的输入源，然后依次为 5，15，35，60，100 个 PC 的重构结果。

2) 数据 1:第一张照片为没有经过训练的输入源，然后依次为 5，15，35，60，100 个 PC 的重构结果。

3) 数据 2: 第一张照片为没有经过训练的输入源，然后依次为 5，15，35，60，100 个 PC 的重构结果。

4) 数据 3: 第一张照片为没有经过训练的输入源，然后依次为 5，15，35，60，100 个 PC 的重构结果。

5） 可以发现，重构结果虽然有一点点模糊，但是基本上可以看出是本人。

3. 识别演示结果：

输入：

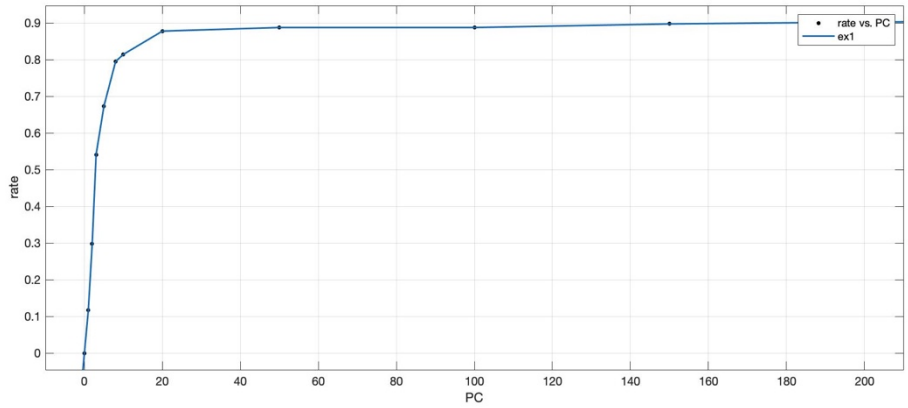
输出：（识别出是第 41 个人，并输出最接近的图片）

4. 训练量为一半的情况下，识别率与 PC 数量的关系：

1） 训练每个人脸的后五张，用前五张进行测试，结果如下：

PC 数	0	1	2	3	5	8	10	20	50	100	150	200
识别率	0	0.1171	0.2976	0.5415	0.6732	0.7951	0.8146	0.8780	0.8878	0.8878	0.8976	0.9024

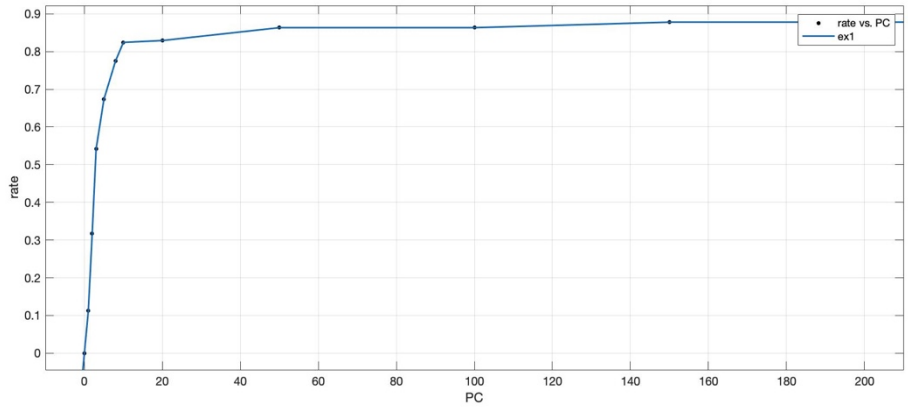
用 matlab 绘图如下：



2） 训练每个人脸的前五张，用后五张进行测试，结果如下：

PC 数	0	1	2	3	5	8	10	20	50	100	150	200
识别率	0	0.1122	0.3171	0.5415	0.6732	0.7756	0.8244	0.8293	0.8634	0.8634	0.8780	0.8780

用 matlab 绘图如下：



6） 结论：

a） 随着 PC 数的增加，识别率不断提高。

b) 除了在个别 PC 数的情况下，第一个实验的识别率一般会稍微高于第二个实验。观察数据库中的照片，发现一般前五张照片都比较“端正”，后五张照片会有点“歪”。因此，我觉得这样的区别产生是因为训练样本不同，导致了训练出来的转换矩阵有轻微的差别导致的。

五、 结论与心得体会

在这一次的实验中，我实现了 EigenFace 并做了多组实验，实现效果比较好，过程中间也踩了不少的坑，总结如下：

1) 几个归一化的地方：用 SVD 法求解出特征向量之后，要进行 NORM_L1 的归一化；重构的时候，取出特征脸，要进行 0～255 的 NORM_MINMAX 归一化；重构完成，也要进行 0～255 的 NORM_MINMAX 归一化。
2) 对于一般读进来的照片，都是三通道的，要么直接用 cvtColor 函数转变成单通道的，要么在用指针调用的时候要注意调用地址需要乘 3。
3) 在构造数据库的时候，每张人脸要减掉平均脸，然后乘上转换矩阵 A；在识别的时候，对于输入的人脸，要减掉平均脸，然后乘上转换矩阵 A，找到在 n 维空间中最接近的点即可。
4) 之前自己习惯了使用 Eigen 库的矩阵类型，因此一开始混用了 Matrix 和 Mat，这是一个不好的习惯，导致后来改了很久，甚至重新把类里面的数据类型全部改掉了，浪费了很多时间。

希望在接下来的课程和作业中，能够接触到更多实用的算法，同时学习到更多关于计算机视觉的理论知识，尝试自己实现更多有趣的功能。