浙江大学

本科实验报告

Eigenface

课程名称：计算机视觉

姓　　名：秦雨扬

学　　院：竺可桢学院

专　　业：自动化（控制）

学　　号：3210104591

电　　话：13588789194

邮　　件：qinyuyang2003@zju.edu.cn

指导老师：潘纲

2023 年 12 月 17 日

Contents

List of Figures

浙江大学实验报告

专业：自动化（控制）
姓名：秦雨扬
学号：3210104591
日期：2023 年 12 月 17 日
地点：玉泉曹光彪西-103

课程名称：计算机视觉　指导老师：潘纲　助教：周健均
实验名称：Eigenface　实验类型：综合　成绩：

I 功能简述及运行说明

1. 功能简述

在本实验中，主要实现了以下几个功能

- 对于给定文件夹，自动遍历其中图片，实现使用鼠标先点左眼再点右眼，可自动生成对应格式的同名标注数据，存放在指定文件夹下
- 根据人眼模版，将所有的人脸进行对齐（自动处理 RGB 和 GRAY 图像），并对所得灰度图像进行直方图均衡化，并将结果保存在指定文件夹中与原文件夹中相同位置的地方
- 将预处理过的人脸库重新导入，并根据百分比将数据集随机地切分为训练集与数据集
- 对训练集中的图像进行 PCA，得到均值脸与特征脸，并与原始数据一同保存在 model 中
- 导入 model, 将输入图像分解到各个特征脸上，并在训练集中找到与该特征欧氏距离最近的点，作为预测值
- 对给定图像进行重构，将其 10 个 PCs、25 个 PCs、50 个 PCs、以及 100 个 PCs 重构的结果拼接在一张图上。
- 对于不同的 PC, 对整个测试集做预测，得到 Rank1-PC 图像，使用 opencv 绘制，并将数据导出 yml 并使用 python 重新画了一遍

2. 运行说明

本实验为结构较为复杂，main.cpp 为主程序，mark_eye.cpp 为人眼位置标注程序，correct_with_eye.cpp 为模版匹配与直方图均衡化的预处理程序，plot.py 则为 Rank1-PC 图像绘制程序，mark_eye.h 与 correct_with_eye.h 为对应的头文件。以下从 main.cpp 中的每个函数具体展开介绍

- mark_eye(input_path,output_path) 为人眼标注函数，这里用来标注 att-face 下 s0 与 s41 两个文件夹中我的两组自定义图像，由于其中使用 imshow 与鼠标响应，此处注释以方便助教运行
- prepare_all(image_path,json_path,output_path) 为模版匹配与直方图均衡化的预处理函数
- load_dataset 为导入数据集，其中第一个值为 train 的占比，根据此百分比将数据集随机地切分为训练集与数据集
- eigenface_train 为训练函数，对训练集中的图像进行 PCA，得到均值脸与特征脸，并与原始数据一同保存在 model 中
- eigenface_reconstruct 为对给定图像进行重构
- void eigenface_test(std::string img_path,std::string model_path,std::string output_path=NULL) 为对单个图像的测试, 而 float eigenface_test(std::string model_path,std::vector<std::pair<Mat,int» test,int PCA_num) 则是对整个测试集进行测试并输出正确率
- test_all() 函数则对 Rank1-PC 进行了测试。

II 开发与运行环境

整体程序采用 c++ 完成, 采用 opencv 4.2.0, 测试系统为 *Ubuntu 20.04.6 LTS*，需要使用 -std=c++17, 同时我本地 opencv2 在 opencv4 文件夹下，所以对 Makefile 有所需改。

对于 Rank1-PC 绘图部分实验采用 python 语言，测试系统为 *Ubuntu 20.04.6 LTS*, 测试环境为 *Anaconda 23.7.3* 下的 *python 3.8.18* ，使用的 numpy 与 matplotlib 包。

III 算法原理与具体实现

1. 数据集生成与标注

对于采集的图像，我们将它放在同一个文件夹下，使用 mark_eye 进行人眼标注，采用 mouse 回调函数进行处理，具体代码实现如下

```cpp
#include"mark_eye.h"

int now=0;
FILE* Files;


static void onMouse( int event, int x, int y, int, void* ){
    if( event != EVENT_LBUTTONDOWN )
        return;
    if(now==0){fprintf(Files,"{\n  \"centre_of_left_eye\": [\n    %d,\n    %d\n   ],\n",x,y);}
    else{
    fprintf(Files,"   \"centre_of_right_eye\": [\n    %d,\n    %d\n   ]\n}",x,y);}
    // circle(img,(447,63), 63, (0,0,255), -1)
    now+=1;
    // std::cout << x << " " << y << std::endl;
}
int mark_eye(std::string input_path/* = "att-face/s1"*/,std::string output_path /*=
 ↪  "ATT-eye-location/s0/"*/)
{
    // std::string image_path = "att-face/s1/1.pgm";
    // std::string output_path = "ATT-eye-location/s0/";
    fs::create_directories(output_path);
    for (const auto& dirEntry : recursive_directory_iterator(input_path)){
        printf("Processing File %s ...\n",dirEntry.path().c_str());
        std::string name=fs::path(dirEntry).stem();
        std::string output_name = output_path+name+".json";
        Files=fopen(output_name.c_str(),"w");

        now=0;
        Mat img = imread(dirEntry.path().string());
        // print(img);
        namedWindow( "Display frame",WINDOW_NORMAL);
        resizeWindow ("Display frame", 900, 1200);
        imshow("Display frame", img);
        setMouseCallback("Display frame",onMouse);
        waitKey(0); // Wait for a keystroke in the window
```

```
        // while(now<2){
        //     waitKey(0);
        // }
        destroyAllWindows();
        fclose(Files);
    }
    // Mat img = imread(image_path);
    // // print(img);
    // namedWindow( "Display frame",WINDOW_NORMAL);
    // resizeWindow ("Display frame", 900, 1200);
    // imshow("Display frame", img);
    // setMouseCallback("Display frame",onMouse);
    // int k = waitKey(0); // Wait for a keystroke in the window
    return 0;
}
```

2. 数据预处理

$$\begin{bmatrix} x* \\ y* \\ 1 \end{bmatrix} = \begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ y_1 \\ 1 \end{bmatrix}$$

通过两对数据点，只有四个方程，但是旋转矩阵有 6 个未知量，这该如何处理?

我最简单的想法就是

$$\begin{bmatrix} x* \\ y* \end{bmatrix} = \begin{bmatrix} h_{11} & h_{12} \\ h_{21} & h_{22} \end{bmatrix} \begin{bmatrix} x_1 \\ y_1 \end{bmatrix}$$

得到 **Figure** 1 的实验结果

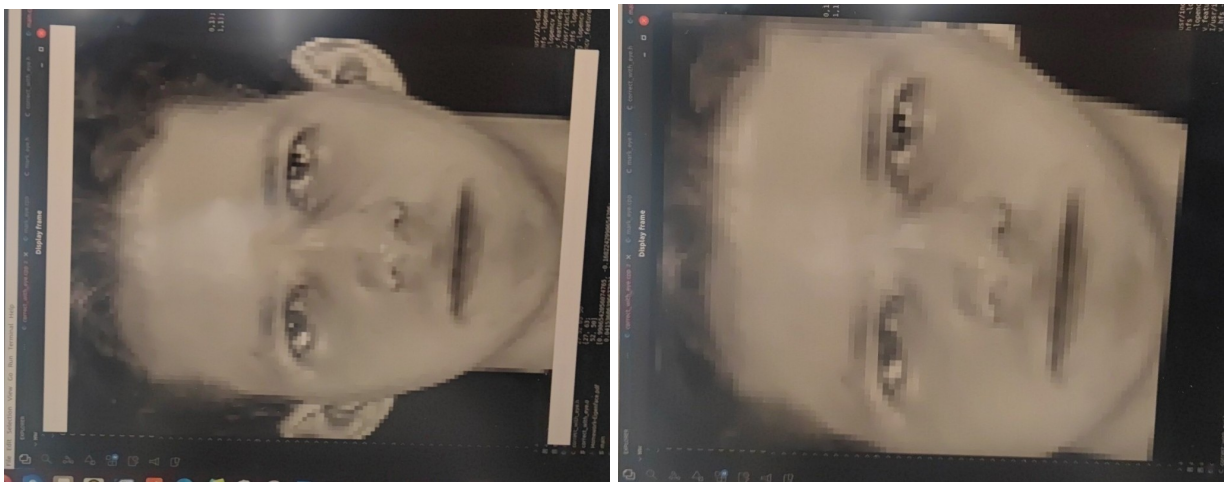

Figure 1: 简化方程实验结果 (不知道为什么转了 90 度)

说明该方法不可行。

后来想到，能不能只旋转，将两只眼睛对齐，如是确定了一个 θ, 以及其对应的旋转矩阵

接着想，能不能让眼睛两对点眼睛的中心对齐？似乎可以仅仅做一个平移实现，

最后想，能不能以两眼睛的中心做一个缩放？可以！
于是乎就完成了整个模版匹配。代码如下

```
int SIZE_l=96,SIZE_w=72,dx1=18,dx2=54,dy1=40,dy2=40;

Mat correct_with_eye(Mat img,int x1,int y1,int x2,int y2)
{
    double dx,dy,dz;
    dx=x2-x1;dy=y2-y1;dz=sqrt(dx*dx+dy*dy);
    Mat middle_point=Mat::zeros(3,1,CV_64F);
    middle_point.at<double>(0,0)=(x1+x2)/2.0;
    middle_point.at<double>(1,0)=(y1+y2)/2.0;
    middle_point.at<double>(2,0)=1;

    Mat srcp=Mat::zeros(3,2,CV_64F);
    srcp.at<double>(0,0)=x1;srcp.at<double>(0,1)=x2;srcp.at<double>(2,0)=1;
    srcp.at<double>(1,0)=y1;srcp.at<double>(1,1)=y2;srcp.at<double>(2,1)=1;
    // std::cout << srcp << std::endl;

    Mat M=Mat::zeros(2,3,CV_64F);
    M.at<double>(0,0)=dx/dz;M.at<double>(0,1)=dy/dz;
    M.at<double>(1,0)=-dy/dz;M.at<double>(1,1)=dx/dz;
    // std::cout << M << std::endl;

    Mat new_middle=M*middle_point;
    // M.at<double>(0,2)=(dx1+dx2)/2.0-new_middle.at<double>(0,0);
    // M.at<double>(1,2)=(dy1+dy2)/2.0-new_middle.at<double>(1,0);
    M.at<double>(0,2)=-new_middle.at<double>(0,0);
    M.at<double>(1,2)=-new_middle.at<double>(1,0);
    M*=sqrt(1.0f*(dx1-dx2)*(dx1-dx2)+1.0f*(dy1-dy2)*(dy1-dy2))/dz;
    M.at<double>(0,2)+=(dx1+dx2)/2;
    M.at<double>(1,2)+=(dy1+dy2)/2;
    // std::cout << M << std::endl;
    // Mat final_middle1=M*middle_point;
    // std::cout << final_middle1 << std::endl;
    // Mat final_middle=M*srcp;
    // std::cout << final_middle << std::endl;
    // Mat M=Mat::zeros(2,3,CV_64F);
    // M.at<double>(0,0)=M0.at<double>(0,0);M.at<double>(0,1)=M0.at<double>(0,1);
    // M.at<double>(1,0)=M0.at<double>(1,0);M.at<double>(1,1)=M0.at<double>(1,1);
    Mat img_dst0=Mat::zeros(SIZE_l,SIZE_w,img.type());
    warpAffine(img,img_dst0,M,Size(SIZE_w,SIZE_l));
    Mat img_dst;
    cvtColor( img_dst0, img_dst0, COLOR_BGR2GRAY );
    // img_dst0.convertTo(img_dst1, CV_8U);
    // std::cout<<"!"<<img_dst1.type()<<std::endl;
    equalizeHist( img_dst0, img_dst );
    // namedWindow( "Display frame",WINDOW_NORMAL);
    // resizeWindow ("Display frame", 900, 1200);
```

```
    // imshow("Display frame", img);
    // waitKey(0);
    // imshow("Display frame", img_dst);
    // waitKey(0);
    // img_dst=img_dst.reshape(1,1);
    // std::cout << img_dst.size() << std::endl;
    return img_dst;
}
```

当然上面那里有一个小插曲，当时 (x1+x2)/2 没注意，一直有精度问题，后来发现它是一个整数除法，属实是 python 用多了

转念一想，那岂不是三对点？就又试了 getAffineTransform，发现其返回的矩阵为全 0 阵，可能需要不在同一直线上。

```
Mat correct_with_eye_test(Mat img,int x1,int y1,int x2,int y2){
    Point2f srcTri[3];
    srcTri[0] = Point2f( x1, y1 );
    srcTri[1] = Point2f( (x1+x2)/2.0 ,(y1+y2)/2.0 );
    srcTri[2] = Point2f( x2, y2 );
    Point2f dstTri[3];
    dstTri[0] = Point2f( dx1,dy1 );
    dstTri[1] = Point2f( (dx1+dx2)/2.0 ,(dy1+dy2)/2.0 );
    dstTri[2] = Point2f( dx2,dy2 );
    Mat M = getAffineTransform( srcTri, dstTri );
    Mat img_dst=Mat::zeros(SIZE_l,SIZE_w,img.type());
    std::cout << M<< std::endl;
    warpAffine(img,img_dst,M,Size(SIZE_w,SIZE_l));
    namedWindow( "Display frame",WINDOW_NORMAL);
    resizeWindow ("Display frame", 900, 1200);
    imshow("Display frame", img);
    waitKey(0);
    imshow("Display frame", img_dst);
    waitKey(0);
    return img_dst;
}
```

其它的数据预处理如下

```
#include "mark_eye.h"
#include "correct_with_eye.h"

int new_shape[] ={1,1};
void prepare_one(std::string image_path,std::string json_path,std::string output_path){
    // std::string image_path = "att-face/s1/1.pgm";
    std::cout<<image_path<<" "<<json_path<<" "<<output_path<<std::endl;
    Mat img=imread(image_path);
    // std::string json_path = "ATT-eye-location/s1/1.json";
    FILE* Files;
```

```
    Files=fopen(json_path.c_str(),"r");
    int x[4],idx=0;
    char ch;
    x[0]=x[1]=x[2]=x[3]=0;
    while(!feof(Files)){
        fread(&ch,1,1,Files);
        if(ch<'0'||ch>'9'){
            if(x[idx]!=0){
                idx+=1;
            }
            continue;
        }
        x[idx]=x[idx]*10+ch-'0';
    }
    // std::cout<<x[0]<<" "<<x[1]<<" "<<x[2]<<" "<<x[3]<<std::endl;
    Mat img_out;
    img_out=correct_with_eye(img,x[0],x[1],x[2],x[3]);
    imwrite(output_path,img_out);
}
void prepare_all(std::string image_path="att-face",std::string json_path="ATT-eye-location",std::string
 ↪  output_path="processed_image"){
    for (const auto& dirEntry : recursive_directory_iterator(image_path)){
        if(fs::is_directory(dirEntry)){
            std::string middle=fs::path(dirEntry).relative_path().string().substr(image_path.length()+1);
            fs::create_directories(output_path+"/"+middle);
            continue;
        }
        printf("Processing File %s ...\n",dirEntry.path().c_str());
        std::string name=fs::path(dirEntry).stem();
        std::string
         ↪  middle=fs::path(dirEntry).relative_path().remove_filename().string().substr(image_path.length()+1);
        // std::string json_name = output_path+name+".json";
        //
         ↪  std::cout<<fs::path(dirEntry).relative_path().remove_filename().string().substr(image_path.length()+1)<<"
         ↪  "<<output_name<<std::endl;

         ↪  prepare_one(dirEntry.path().string(),json_path+"/"+middle+name+".json",output_path+"/"+middle+name+".pgm")
        // break;
    }
    // correct_with_eye(image_path,json_path);
}
```

3. 特征脸提取

Let $X = \{x_1, x_2, \dots, x_n\}$ be a random vector with observations $x_i \in R^d$.
Compute the mean μ

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

Compute the the Covariance Matrix S

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

Compute the eigenvalues λ_i and eigenvectors v_i of S

$$Sv_i = \lambda_i v_i, i = 1, 2, \ldots, n$$

Order the eigenvectors descending by their eigenvalue. The k principal components are the eigenvectors corresponding to the k largest eigenvalues.

The k principal components of the observed vector x are then given by:

$$y = W^T(x - \mu)$$

where $W = (v_1, v_2, \ldots, v_k)$.

The reconstruction from the PCA basis is given by:

$$x = Wy + \mu$$

where $W = (v_1, v_2, \ldots, v_k)$.

The Eigenfaces method then performs face recognition by: - Projecting all training samples into the PCA subspace. - Projecting the query image into the PCA subspace. - Finding the nearest neighbor between the projected training images and the projected query image.

Still there's one problem left to solve. Imagine we are given 400 images sized 100×100 pixel. The Principal Component Analysis solves the covariance matrix $S = XX^T$, where size$(X) = 10000 \times 400$ in our example. You would end up with a 10000×10000 matrix, roughly $0.8GB$. Solving this problem isn't feasible, so we'll need to apply a trick. From your linear algebra lessons you know that a $M \times N$ matrix with $M > N$ can only have $N-1$ non-zero eigenvalues. So it's possible to take the eigenvalue decomposition $S = X^T X$ of size $N \times N$ instead:

$$X^T X v_i = \lambda_i vi$$

and get the original eigenvectors of $S = XX^T$ with a left multiplication of the data matrix:

$$XX^T(Xv_i) = \lambda_i(Xv_i)$$

The resulting eigenvectors are orthogonal, to get orthonormal eigenvectors they need to be normalized to unit length. I don't want to turn this into a publication, so please look into [60] for the derivation and proof of the equations.

以上引用自 Opencv 官网文档 [1]，主要的思想是去求 $N \times N$ 矩阵 $S = X^T X$ 的特征值，而不是直接对 $S = XX^T$ 进行分解。

其余的部分个人感觉就挺顺理成章的，主要是代码实现：

只要保持特征向量模长为 1，reconstruct 就是求个点积得到投影，再根据投影长度把所有主成分进行加权组合就可以拟合原向量

test 则是利用了这个数据降维的过程，通过欧氏距离下的最近邻来进行识别。

实现程序如下

```
47  void load_dataset(float split,std::string path,std::vector<std::pair<Mat,int>>
    ↪  &train,std::vector<std::pair<Mat,int>> &test){
48      int num=int(split*10);
49      int a[]={1,2,3,4,5,6,7,8,9,10};
50      // std::vector<pair<Mat,int>> train,test;
51      for(int i=0;i<=41;i++){
52          std::random_shuffle(a,a+10);
53          for(int j=0;j<num;j++){
54              int x=a[j];
55              std::string name=path+"/s"+std::to_string(i)+"/"+std::to_string(x)+".pgm";
56              // std::cout<<name<<std::endl;
57              Mat img=imread(name,cv::IMREAD_GRAYSCALE);
58              // new_shape=img.size();
59              new_shape[0]=img.rows;
60              new_shape[1]=img.cols;
61              // std::cout<<SIZE_X<<" "<<SIZE_Y<<std::endl;
62              img=img.reshape(1,1);
63              img.convertTo(img, CV_64F);
64              train.push_back(std::make_pair(img,i));
65          }
66          for(int j=num;j<10;j++){
67              int x=a[j];
68              std::string name=path+"/s"+std::to_string(i)+"/"+std::to_string(x)+".pgm";
69              // std::cout<<name<<std::endl;
70              Mat img=imread(name,cv::IMREAD_GRAYSCALE).reshape(1,1);
71              img.convertTo(img, CV_64F);
72              test.push_back(std::make_pair(img,i));
73          }
74      }
75      // return train,test;
76  }
77  Mat vec2img(Mat v){
78      // std::cout<<v.size()<<std::endl;
79      // std::cout<<SIZE_X<<" "<<SIZE_Y<<std::endl;
80      // vector<int> new_shape={SIZE_X,SIZE_Y};
81      v=v.reshape(1,2,new_shape);
82      v.convertTo(v, CV_8U);
83      // imshow("Display frame", v);
84      // waitKey(0);
85      return v;
86  }
87
88  void eigenface_test(std::string img_path,std::string model_path,std::string output_path=NULL){
89      Mat img=imread(img_path,cv::IMREAD_GRAYSCALE);
90      img=img.reshape(1,1);
91      img.convertTo(img, CV_64F);
92      cv::FileStorage file(model_path, cv::FileStorage::READ);
93      Mat mean,u,data,X;
94      std::vector<int> labels;
```

```
    file["mean"] >> mean;
    file["u"] >> u;
    file["labels"] >> labels;
    file["data"] >> data;
    file["X"] >> X;

    Mat img_show=vec2img(img);
    img=img-mean;
    Mat tmp=u*img.t();
    float last_sum=-1,sum=0;
    int last_idx=0;
    // Mat new_img=Mat::zeros(1,img.cols,CV_64F);
    for(int i=0;i<data.cols;i++){
        sum=0;
        for(int j=0;j<data.rows;j++){
            sum+=(data.at<double>(j,i)-tmp.at<double>(j,0))*(data.at<double>(j,i)-tmp.at<double>(j,0));
        }
        if(last_sum==-1||sum<last_sum){
            last_sum=sum;
            last_idx=i;
        }
    }
    Mat nearset=X(Range(last_idx,last_idx+1),Range(0,X.cols))+mean;
    std::cout<<img.size()<<" "<<nearset.size()<<std::endl;
    // return;
    hconcat(img_show,vec2img(nearset),img_show);
    cvtColor(img_show,img_show,COLOR_GRAY2BGR);
    putText(img_show, //target image
            img_path.substr(16, 5), //text
            cv::Point(5, img_show.rows-5), //top-left position
            cv::FONT_HERSHEY_DUPLEX,
            1.0,
            CV_RGB(118, 185, 0), //font color
            2);
    putText(img_show, //target image
            "s"+std::to_string(labels[last_idx]), //text
            cv::Point(new_shape[1]+5, img_show.rows-5), //top-left position
            cv::FONT_HERSHEY_DUPLEX,
            1.0,
            CV_RGB(255, 0, 0), //font color
            2);
    imwrite(output_path,img_show);
}
void eigenface_reconstruct(std::string img_path,std::string model_path,std::string
 ↪  output_path="test.jpg"){
    Mat img=imread(img_path,cv::IMREAD_GRAYSCALE);
    img=img.reshape(1,1);
    img.convertTo(img, CV_64F);
    cv::FileStorage file(model_path, cv::FileStorage::READ);
```

```
    Mat mean,u;
    std::vector<int> labels;
    file["mean"] >> mean;
    file["u"] >> u;
    file["labels"] >> labels;
    file.release();

    Mat img_show=vec2img(img);
    img=img-mean;
    Mat tmp=u*img.t();
    // Mat new_img=Mat::zeros(1,img.cols,CV_64F);
    for(int i=0;i<tmp.rows;i++){
        if(i==10||i==25||i==50||i==100){
            // vec2img(mean);
            hconcat(img_show,vec2img(mean),img_show);
        }
        mean+=tmp.at<double>(i,0)*u.row(i);
    }
    imwrite(output_path,img_show);
    // std::cout<<tmp.size()<<std::endl;
    // std::cout<<tmp<<std::endl;
}
void eigenface_train(float energy,std::string output_path,std::vector<std::pair<Mat,int>> &train){
    float n=train.size();
    int PCA_num=int((energy/100*n));
    int m=train[0].first.cols;
    std::cout<<n<<" "<<m<<std::endl;

    Mat mean=Mat::zeros(1,m,CV_64F);
    for(auto i : train){
        mean+=i.first;
    }
    mean=mean/n;
    Mat X;
    std::vector<int> labels;
    for(auto i : train){
        if(X.empty()){
            X=i.first-mean;
        }else{
            vconcat(X,i.first-mean,X);
        }
        labels.push_back(i.second);
    }
    Mat S=X*X.t();
    Mat  w, u, vt;
    SVD::compute(S, w, u, vt);
    // std::cout<<w<<std::endl;
    // std::cout<<u*u.t()<<std::endl;
    u=X.t()*u;
```

```
192
193     u=u.t();
194     int n0=u.rows,m0=u.cols;
195     double sum=0,max=0;
196     std::cout<<n0<<" "<<m0<<std::endl;
197     u=u(Range(0,PCA_num),Range(0,m0));
198     n0=u.rows,m0=u.cols;
199     std::cout<<n0<<" "<<m0<<std::endl;
200     Mat eigenface_show=vec2img(mean);
201     for(int i=0;i<n0;i++){
202         // vec2img(u.row(i)/378*255);
203         sum=0;
204         max=0;
205         for(int j=0;j<m0;j++){
206             sum+=u.at<double>(i,j)*u.at<double>(i,j);
207         }
208         // std::cout<<sum<<std::endl;
209         sum=sqrt(sum);
210         // if(sum<1e-8){
211         //     continue;
212         // }
213         for(int j=0;j<m0;j++){
214             u.at<double>(i,j)/=sum;
215             if(abs(u.at<double>(i,j))>max){
216                 max=abs(u.at<double>(i,j));
217             }
218         }
219         if(i<10)
220             hconcat(eigenface_show,vec2img(u.row(i)/max*255),eigenface_show);
221     }
222     Mat data=u*X.t();
223     imwrite(output_path+"_eigenface.jpg",eigenface_show);
224
225     cv::FileStorage file(output_path, cv::FileStorage::WRITE);
226     // cv::Mat someMatrixOfAnyType;
227
228     // Write to file!
229     file << "mean" << mean << "u" << u << "labels" << labels<<"data"<<data<<"X"<<X;
230
231     // Close the file and release all the memory buffers
232     file.release();
233 }
234
235 float eigenface_test(std::string model_path,std::vector<std::pair<Mat,int>> test,int PCA_num){
236     cv::FileStorage file(model_path, cv::FileStorage::READ);
237     Mat mean,u,data,X;
238     std::vector<int> labels;
239     file["mean"] >> mean;
240     file["u"] >> u;
```

```
    file["labels"] >> labels;
    file["data"] >> data;
    file["X"] >> X;
    file.release();

    // int PCA_num=int((energy/100*test.size()));
    int n0=u.rows,m0=u.cols;
    u=u(Range(0,PCA_num),Range(0,m0));
    n0=u.rows,m0=u.cols;
    // std::cout<<n0<<" "<<m0<<std::endl;
    Mat img;
    float correct=0;
    for(auto xk: test){
        // important clone
        img=xk.first.clone();
        img=img-mean;
        Mat tmp=u*img.t();
        // std::cout<<tmp<<std::endl;
        // return 0;
        double last_sum=-1,sum=0;
        int last_idx=0;
        // Mat new_img=Mat::zeros(1,img.cols,CV_64F);
        for(int i=0;i<data.cols;i++){
            sum=0;
            // std::cout<<data.rows<<" "<<tmp.rows<<std::endl;
            for(int j=0;j<tmp.rows;j++){
                // std::cout<<data.at<double>(j,i)<<" "<<tmp.at<double>(j,0)<<" |";
                sum+=(data.at<double>(j,i)-tmp.at<double>(j,0))*(data.at<double>(j,i)-tmp.at<double>(j,0));
                // std::cout<<sum<<" ";
            }
            //  std::cout<<sum<<" ";
            if(last_sum<0||sum<last_sum){
                last_sum=sum;
                last_idx=i;
            }
            // if(i>3) break;
        }
        // std::cout<<last_sum<<" "<<xk.second<<" "<<last_idx<<" "<<labels[last_idx]<<std::endl;
        if(labels[last_idx]==xk.second){
            correct+=1;
        }
        // break;
    }
    // std::cout<<correct<<" "<<test.size()<<std::endl;
    return correct/test.size();
}
void test_all(){
    std::vector<std::pair<Mat,int>> train,test;
```

```
    load_dataset(0.5,"processed_image",train,test);
    eigenface_train(100,"model_test.yml",train);

    // float x=eigenface_test_all("model_test.yml",test,5);
    // std::cout<<x<<std::endl;
    cv::Mat image = cv::Mat::zeros(210, 200, CV_8UC3);
    image.setTo(cv::Scalar(255, 255, 255));
    std::vector<cv::Point> points;
    std::vector<double> points_py;
    std::cout<<"Start Ploting rank1-pc figure"<<std::endl;
    for(int PCA_num=1;PCA_num<=200;PCA_num+=1){
        float x=eigenface_test("model_test.yml",test,PCA_num);
        points.push_back(cv::Point(PCA_num, (1-x)*200));
        points_py.push_back(x);
        // std::cout<<"!"<<PCA_num<<" "<<x<<std::endl;
    }
    cv::FileStorage file("points_py.yml", cv::FileStorage::WRITE);
    // cv::Mat someMatrixOfAnyType;

    // Write to file!
    file << "points" << points_py<<"gap" << 1;

    // Close the file and release all the memory buffers
    file.release();
    // for (int i = 0; i < points.size(); i++){
        //         cv::circle(image, points[i], 5, cv::Scalar(0, 0, 0), 2, 8, 0);
        // }
    cv::polylines(image, points, false, cv::Scalar(0, 255, 0), 1, 8, 0);
    imwrite("test_all.jpg",image);
    // x=eigenface_test("model_test.yml",test,5);
    // std::cout<<x<<std::endl;
    // x=eigenface_test_all("model_test.yml",test,5);
    // std::cout<<x<<std::endl;
}
int main(){
    // mark_eye("att-face/s41","ATT-eye-location/s41/");
    prepare_all();
    std::vector<std::pair<Mat,int>> train,test;
    load_dataset(0.5,"processed_image",train,test);
    eigenface_train(25,"model.yml",train);
    eigenface_reconstruct("processed_image/s1/1.pgm","model.yml","reconstruct.jpg");
    eigenface_test("processed_image/s11/2.pgm","model.yml","test.jpg");
    test_all();
    return 0;
}
```

IV 实验结果与分析

首先来看前 10 平均人脸与至少前 10 个特征脸，如 **Figure** 2所示，这里我因为将特征模长归一化过，故在展示时将其重新映射到 0255 之间，而平均脸则为直接展示。

Figure 2: eigenface

test 函数的实验结果如 **Figure** 3所示，从嘴巴可见其实两张图并非同一张图，但识别效果较好

Figure 3: test

如 **Figure** 4所示，在戴眼镜和不带眼镜的情况下我的重构图像均表现较好

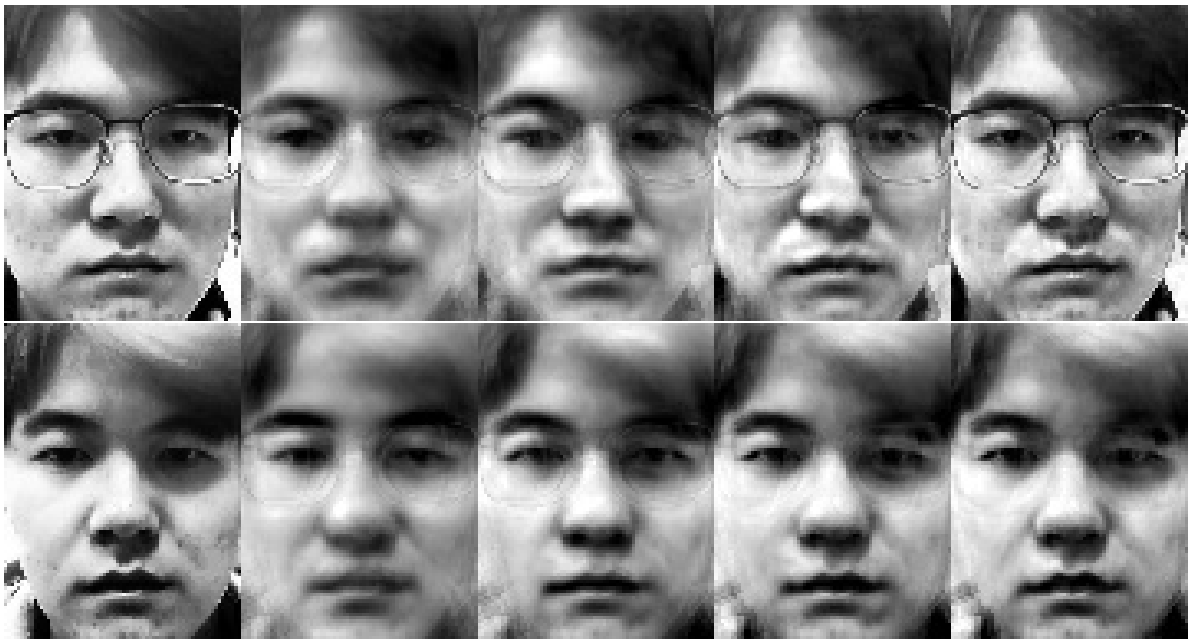

Figure 4: reconstruct

而上方图像的原图如 **Figure** 5所示，可见眼睛的标定与模版匹配均非常成功。

Figure 5: 眼睛模版对齐

最后的 Rank-1 rate-PC num 图像如如 **Figure** 6所示

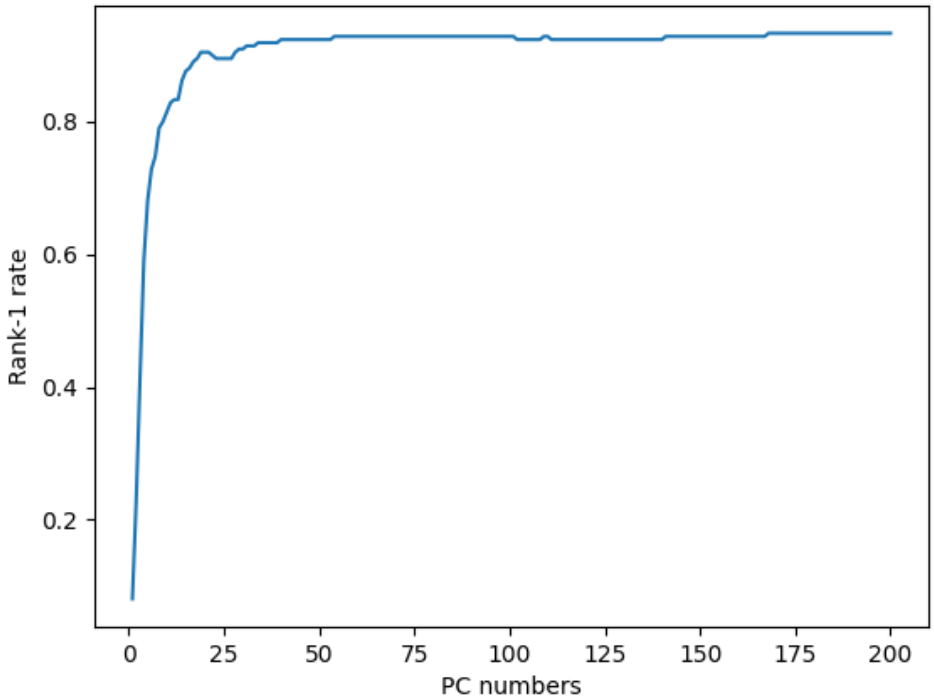

Figure 6: Rank-1 rate-PC number

V 心得体会

- 之前的实验都做的有些匆忙，这一次从周五开始做，想有一个周末的时间可以研究，决定开始大胆地尝试 c++, 结果发现，这个工程量大了不止一点点，属实是怀念 python 了。而且最令我崩溃的是装了一个下午 opencv 都还没装好。一开始怕搞错还特地从源代码直接编译安装，结果发现没有默认开启 pkg，还得卸了重装。加上 opencv2 不知道为什么会是 opencv4/opencv2, 折腾了我好久。加上不是很看得懂 Makefile, 当时一整个下午加晚上终于能够正常使用 opencv, 真的心态都快炸了。更不必说，后来被 C++ 严格的定义与少得可怜的拓展包弄的欲哭无泪。而好久不写 C++ 一切都那么的陌生。好在 DDL 延了一周，不然感觉我写到凌晨 8 点都不一定能完工。
- 在眼睛位置标定过程中，我使用了 opencv 的 GUI 交互，又重新学习了一下 C++ 的版本。后面模版匹配真的让我非常苦恼。好在助教当时硬是没有告诉我，我终于一点点走向柳暗花明。从想到只旋转，再想到中点对齐，再想到缩放。虽然又是一个下午加晚上，但是那种喜悦真的难以描述。
- 在本实验之前，我一直以为我懂了 PCA。而这个实验真的让我看到了自己的不足。好在后来找到了官方文档，不然我要何时才能想到答案。不过这一次，算是真的懂了。
- 虽然本实验花费了非常多的时间，也完成了 500 多行的代码，但是重新去学习 C++，并去不断挑战自己的极限，真的让我也受益匪浅。也感谢助教的努力付出，让我相信我的努力是会被看到的，让我更加有认真完成的动力。

参考文献

[1]https://docs.opencv.org/3.4/da/d60/tutorial_face_main.html