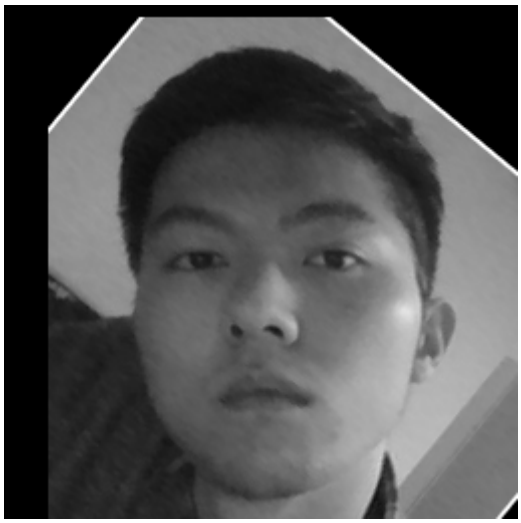
人脸识别

训练过程

数据预处理

本次实验使用的是ftp上的人脸数据，注意到这份数据是已经完成了人脸对齐的数据，所以我们额外需要的工作就是添加自己人脸的数据并且进行仿射变换和人脸对齐。 首先，需要把输入人脸的眼睛定位出来，我是用了opencv自带的眼睛检测器进行自动的识别，返回给我眼睛的坐标。但是这个识别器并不是准确的，他对于很多图都无法准确定位眼睛位置，所以最好的办法还是人工标坐标。 之后，对于输入图形进行旋转，使得两只眼睛能够水平放置。完成后的样例如下：

考虑到自己图片有大量干扰的背景信息，而数据集中数据的背景信息没用但是却占据大量像素（也意味着训练时占用大量计算资源）， 我们最后对于图片进行了切割，只得到了人脸图片，去掉了所有的背景信息。同时因为避免之后求特征向量时耗费大量时间，我们把所有图片resize乘64 * 64的小图片。

训练

1. 将训练集的每张图直方图均衡化后拉伸为一个向量，所有图片组成一个矩阵。

   在这一步要注意每张图的size不能过大，不然训练时间会很长，我使用的是64*64的图片大小。

2. 获得平均脸并且计算出协方差矩阵。下图就是平均脸。

3. 计算协方差矩阵的特征值和特征向量， 由于协方差矩阵是一个由图片大小决定的巨大矩阵，所以这一步会花费很长的时间。
4. 获得最大的几个特征向量得到一个转换矩阵，保存该矩阵，并记录下部分特征脸。

5. 用该矩阵转换所有在训练集中图减去平均脸的数据，得到每一个图片的特征向量。保存这些向量组成的矩阵。

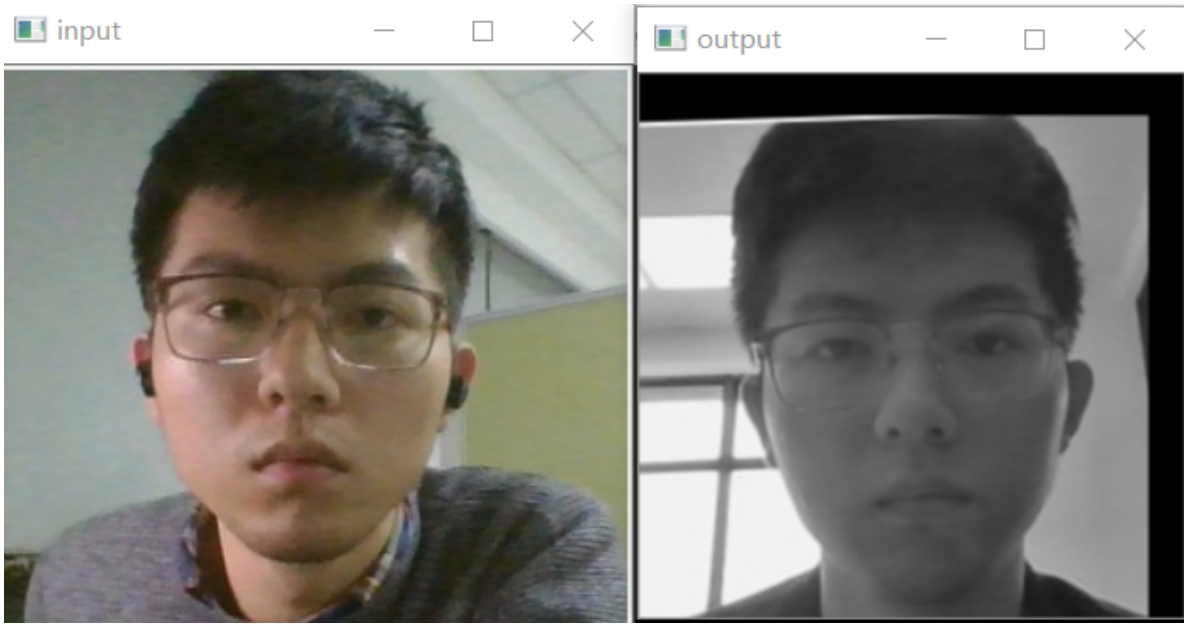
测试过程

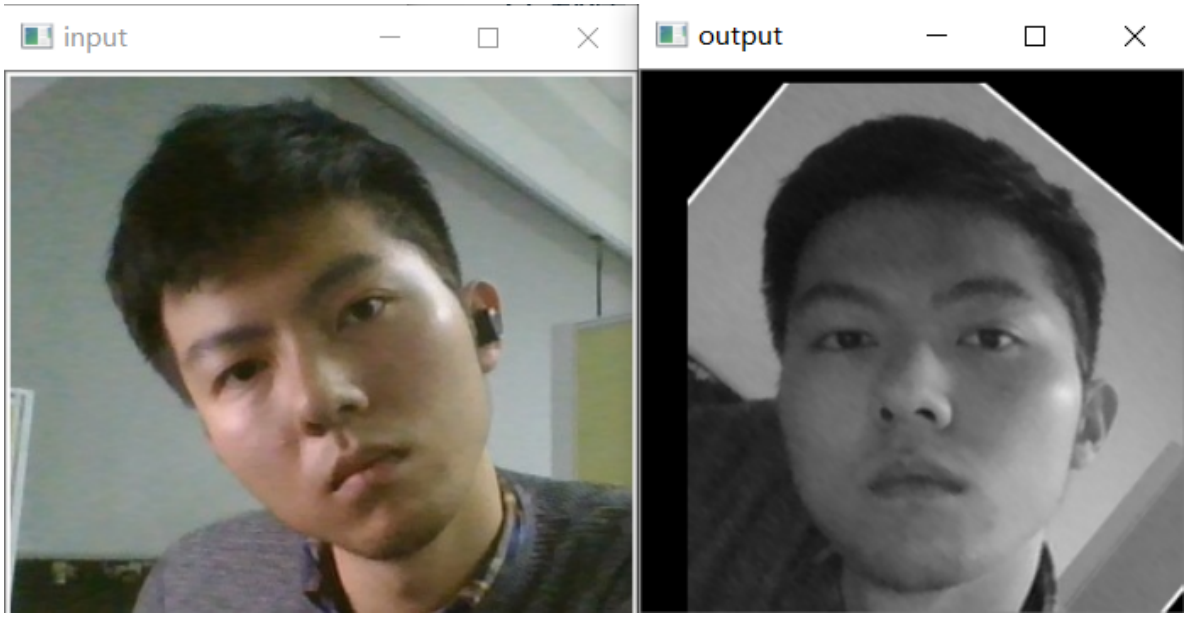
输入一张图片，进行上述的图片预处理过程（由于opencv的眼睛识别的不可靠性，该过程不一定能成功，最保险的办法应该是手动添加好眼睛的坐标然后再处理）。

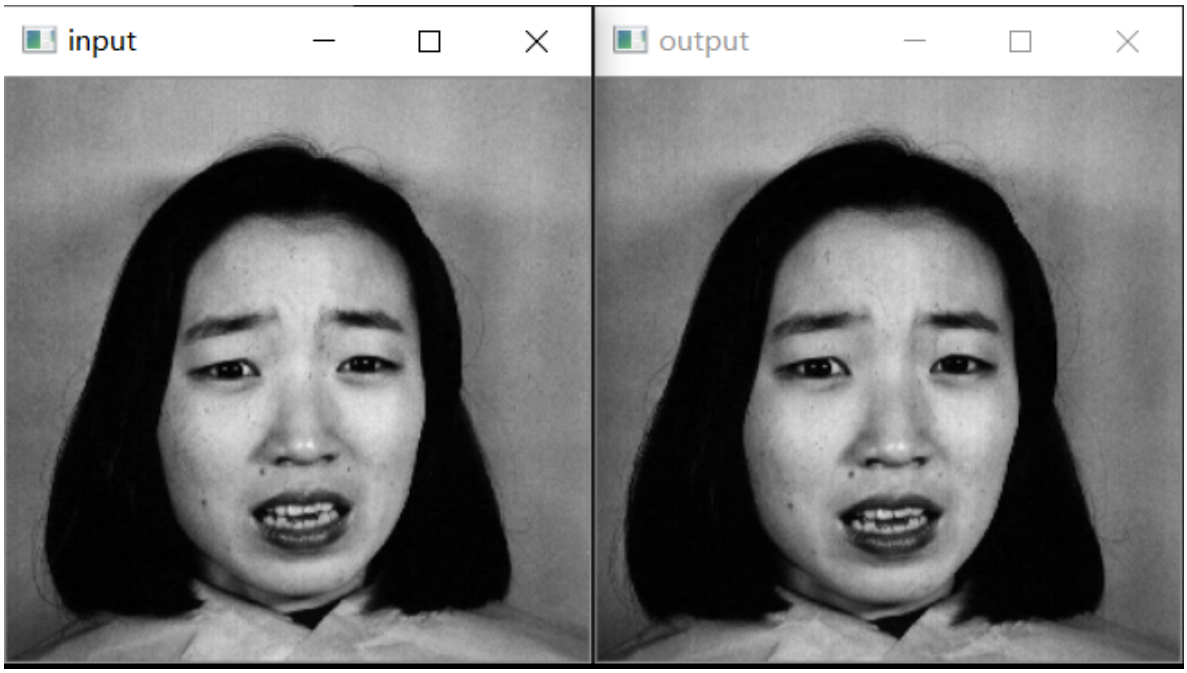
从训练好的结果里读入转换矩阵`A`和数据矩阵`data`。将输入图片`f`通过公式$y_f = A^T(f - meanface)$得到特征向量，将该向量与data矩阵中的所有数据计算欧式距离，距离最近的即为识别结果。

将最近的特征向量重构回图并且加在预处理过的50*50图上得到叠加图。由于叠加图太小，所以把它resize到256 * 256会很模糊。

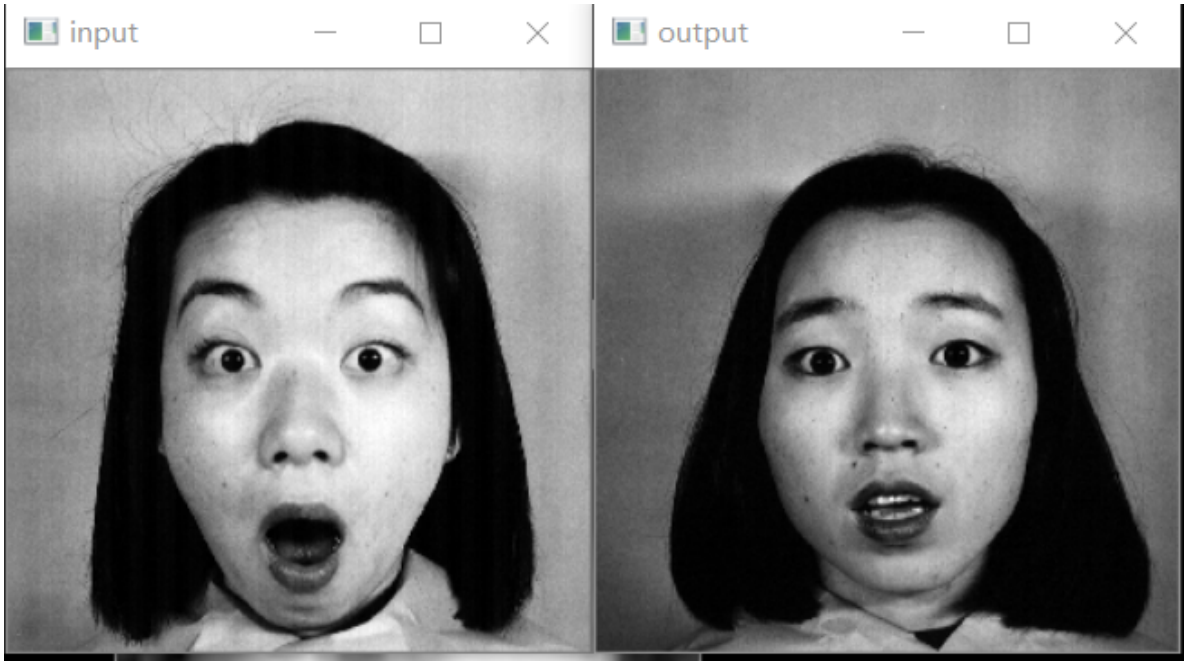
以下使用的输入图片都不在测试集内：







当然，也有识别错误的情况：



在人看来这两个人是不一样的，但是哪怕我已经在图片预处理中做过了裁剪，这个头发还是会干扰到机器对人脸的判断。因此在干扰的情况下，机器会做出误判。

部分关键代码

preprocess

```
Mat preprocess(string dir)
{
    Mat demo = imread("Dataset/coursedata/KA.FE3.47.tiff");
    Mat img, tmp, dst;
    Mat total(IMG_SIZE * IMG_SIZE, 1, CV_8UC1, Scalar(255));
    img = imread(dir );
    imshow("input", img);
    waitKey(10);
    Mat img_align = alignImages(img, demo, dir);
    imshow("alignimg", img_align);
    waitKey(10);
    Rect rect(45, 61, 175, 175);
    img = img_align(rect);
    imshow("img", img);
    waitKey(0);
    cvtColor(img, img, COLOR_RGB2GRAY);
    equalizeHist(img, img);
    resize(img, img, Size(IMG_SIZE, IMG_SIZE));
    tmp = img.reshape(1, IMG_SIZE*IMG_SIZE);
    tmp.col(0).copyTo(total.col(0));
    return total;
}
```

alignimage

```
Mat alignImages(Mat& src, Mat& dst, string dir)
{
    //Mat src_grey, dst_grey;
    //cvtColor(src, src_grey, COLOR_RGB2GRAY);
    //cvtColor(dst, dst_grey, COLOR_RGB2GRAY);
```

```
    Point eyes_src[2];
    dir = dir.substr(0, dir.find_last_of(".")) + ".txt";
    cout << "dir:" << dir << endl;
    ifstream infile;
    infile.open(dir.data());   //将文件流对象与文件连接起来
    assert(infile.is_open());   //若失败,则输出错误消息,并终止程序运行
    string s;
    int i = 0;
    while (getline(infile, s))
    {
        int x, y;
        x = atoi((s.substr(0, s.find_last_of(" ")).c_str()));
        y = atoi((s.substr(s.find_last_of(" ")).c_str()));
        eyes_src[i++] = Point(x, y);
    }
    infile.close();
    Point2f eyesCenter = Point2f( (eyes_src[0].x + eyes_src[1].x)*0.5f,
(eyes_src[0].y + eyes_src[1].y)*0.5f);
    cout << eyes_src[0] << endl;
    cout << eyes_src[1] << endl;
    double dy = eyes_src[1].y - eyes_src[0].y;
    double dx = eyes_src[1].x - eyes_src[0].x;
    double angle = atan2(dy, dx) * 180.0 / CV_PI; // Convert from radians
to degrees.
    Mat rot_mat = getRotationMatrix2D(eyesCenter, angle, 1.0);
    Mat tmp;
    warpAffine(src, tmp, rot_mat, Size(256,256));
    //DetectEye(tmp, eyes_src);
    eyesCenter = Point2f((eyes_src[0].x + eyes_src[1].x)*0.5f,
(eyes_src[0].y + eyes_src[1].y)*0.5f);
    //cout << eyesCenter << endl;
    Point eyes_dst[2];
    eyes_dst[0] = Point(98, 128);
    eyes_dst[1] = Point(160, 128);
    Point2f dst_eyesCenter = Point2f((eyes_dst[0].x + eyes_dst[1].x)*0.5f,
(eyes_dst[0].y + eyes_dst[1].y)*0.5f);
    cout << "dst eyes:"<< eyes_dst[0] <<eyes_dst[1] << endl;
    Mat rot;
    Mat t_mat = cv::Mat::zeros(2, 3, CV_32FC1);
    t_mat.at<float>(0, 0) = 1;
    t_mat.at<float>(0, 2) = dst_eyesCenter.x - eyesCenter.x; //水平平移量
    t_mat.at<float>(1, 1) = 1;
    t_mat.at<float>(1, 2) = dst_eyesCenter.y - eyesCenter.y; //竖直平移量
    warpAffine(tmp, rot, t_mat, Size(256, 256));
    return rot;
}
```

train

```
void train(Mat total, string dir, int numofEigenface, string model)
{
    Mat meanface = GetMeanFace(total);//meanface : IMG_SIZE^2 * 1
    cout << "calculating the Covariance..." << endl;
    Mat Covariance = GetCovariance(total, meanface);//Covariance :
IMG_SIZE^2 * IMG_SIZE^2

```

```
    Mat meanfaceMatrix(IMG_SIZE * IMG_SIZE, total.cols, CV_8UC1, Scalar(255));
    for (int i = 0; i < total.cols; i++)
    {
        meanface.copyTo(meanfaceMatrix.col(i));
    }
    Mat meanfaceMatrix_32F;
    meanfaceMatrix.convertTo(meanfaceMatrix_32F, CV_32FC1);
    Mat engValues, engVectors;
    Mat At( numofEigenface, IMG_SIZE * IMG_SIZE, CV_32FC1);
    /*
    @param eigenvalues output vector of eigenvalues of the same type as src; the eigenvalues are stored
    in the descending order.
    @param eigenvectors output matrix of eigenvectors; it has the same size and type as src; the
    eigenvectors are stored as subsequent matrix rows, in the same order as the corresponding
    eigenvalues.
    */
    cout << "calculating the engValues and engVectors..." << endl;
    eigen(Covariance, engValues, engVectors);
    FileStorage fs(".\\" + model + ".xml", FileStorage::WRITE);
    cout << "get the transform matrix and the max 10 engfaces..." << endl;
    cout << "engValues :" << engValues << endl;
    for (int i = 0; i < numofEigenface; i++)
    {
        if (i < 10) {
            Mat face = engVectors.row(i).clone();
            //cout << face << endl;
            Mat r;
            normalize(face, r, 255.0, 0.0, NORM_MINMAX);
            Mat r_8U;
            r.convertTo(r_8U, CV_8UC1);
            //cout << r << endl;
            imwrite("engface" + to_string(i) + ".jpg", r_8U.reshape(0, IMG_SIZE));
        }
        engVectors.row(i).copyTo(At.row(i));
    }
    fs << "meanface" << meanface;
    fs << "A" << At.t();
    cout << "transorm the images in dataset and save..." << endl;
    Mat total_32F;
    total.convertTo(total_32F, CV_32FC1);
    Mat data = At * (total_32F-meanfaceMatrix_32F);
    fs << "data" << data;
    fs.release();
}
```