

浙江大学

课程名称：计算机视觉

姓 名：刘佳润

学 院：计算机科学与技术学院

专 业：数字媒体技术

学 号：3180105640

指导教师：潘纲

2020 年 12 月 26 日

浙江大学实验报告

课程名称：____计算机视觉____实验类型：____综合____

实验项目名称：____Eigenface 人脸识别____

学生姓名：____刘佳润____专业：____数字媒体技术____学号：____3180105640____

同组学生姓名：____指导老师：____潘纲____

实验地点：____实验日期：____2020____年____12____月____26____日

一、实验内容和要求

自己构建数据库，可选用 AT&T 数据库，包含自己的人脸，进行 eigenface 人脸识别的训练、识别、重构

- 训练：指定能量百分比，将训练结果输出到 model 文件，展示平均脸与前十个特征脸；
- 识别：装载 model 文件，对输入的人脸图像进行识别，将识别结果叠加在输入的人脸图像上，展示训练库中最相似的图像；
- 重构：装载 model 文件，对输入的人脸图像变换到特征脸空间，然后再用变换后的结果重构回原图像。显示自己人脸图像的 10PCs、25PCs、50PCs、100PCs 的重构结果；
- 一半数据做训练，另一半做测试，显示随着 PC 增加，Rank-1 识别率曲线。

二、实验器材

Python 3.7

OpenCV 4.5.0

开发平台：Visual Studio Code

主要使用的 python 开源库有：cv2、numpy、matplotlib

三、具体实现

1. 将自己的人脸加入数据库

调用摄像头，用 opencv 自带的 haar_cascade_frontalface_default.xml 来进行人脸识别，调整框的大小，切出自己的人脸，再 resize 到和 AT&T 数据库一样的数值，转换为 pgm 格式，命名为 s41。

```
faceCascade = cv2.CascadeClassifier('haarcascade_frontalface_default.xml')

def capture():
    """调用摄像头实时截取人脸，输出灰度图"""
    cap = cv2.VideoCapture(0) # 调用摄像头
    cnt = 1
    while(True):
        ret, frame = cap.read()
        gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY) # 输出灰度图
        faces = faceCascade.detectMultiScale(gray, 1.3, 5, minSize = (100,100)) # 检测人脸
        for (x, y, w, h) in faces:
            cv2.rectangle(frame, (x-5, y-30), (x + w+5, y + h+20), (255, 0, 0), 1) # 调整框体大小，与ORL数据库基本相仿
            roiImg = frame[y-30:y+h+20,x-5:x+w+5]
            gray = cv2.cvtColor(roiImg, cv2.COLOR_BGR2GRAY)
            cv2.imwrite('raw/r{}.png'.format(cnt), gray)
            cnt += 1
        cv2.imshow("camera",frame)
        if cv2.waitKey(1) & 0xFF == ord('q'):
            break
    cap.release()
    cv2.destroyAllWindows()

def handle():
    """resize图片，转换到pgm格式"""
    for i in range(10):
        img = cv2.imread('s41/r' + str(i+1) + '.png')
        new_img = cv2.resize(img, (92, 112))
        cv2.imwrite('s41/{}.png'.format(i+1), new_img)
        Image.open('s41/{}.png'.format(i+1)).convert('L').save('s41/{}.pgm'.format(i+1))
```

fig1.1 预处理部分代码

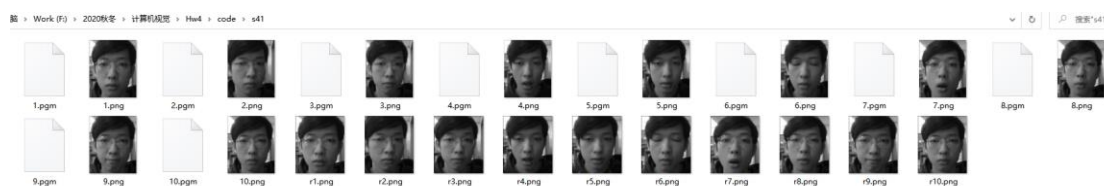


fig1.2 处理后的自己人脸图像

2. 训练过程

先使用一半的数据进行训练。数据集集中的每一张图是 112×92 大小的，用 reshape 函数将其化为 10304×1 的列向量，合并得到全数据集的样本矩阵，维度为 10304×205 。

将所有的人脸在对应维度上加和求平均可以得到“平均脸”。将每个训练图

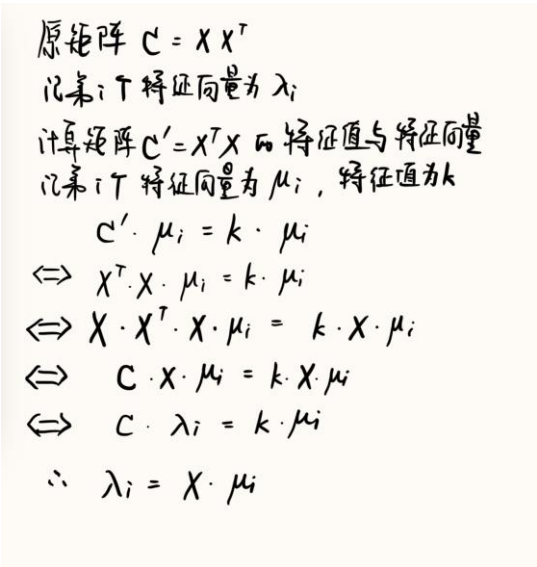
像减去平均图像，可以得到差值图像的数据矩阵。

```
"""
进行数据训练
Param:
FaceMat: 全数据矩阵(205x10304)
energy: 能量 default=0.95
"""
FaceMat = np.mat(FaceMat).T          # 全数据矩阵转置(10304x205)
mean_face = np.mean(FaceMat, 1)      # 平均脸
diffTrain = FaceMat - mean_face      # 得到插值图像的数据矩阵
```

fig2.1 求平均脸与训练数据的差值图像

下一步我们要计算这个矩阵的协方差矩阵。然而偏差值图像的协方差矩阵维度是 10304×10304 ，再求其特征值和特征向量的话，计算量太大。因此这采用一种简单的做法——求一个代替的协方差矩阵，再通过求得特征值与特征向量反推原矩阵的特征向量。

证明如下：



原矩阵 $C = X X^T$
记第 i 个特征向量为 λ_i
计算矩阵 $C' = X^T X$ 的特征值与特征向量
记第 i 个特征向量为 μ_i ，特征值为 k
 $C' \cdot \mu_i = k \cdot \mu_i$
 $\Leftrightarrow X^T \cdot X \cdot \mu_i = k \cdot \mu_i$
 $\Leftrightarrow X \cdot X^T \cdot X \cdot \mu_i = k \cdot X \cdot \mu_i$
 $\Leftrightarrow C \cdot X \cdot \mu_i = k \cdot X \cdot \mu_i$
 $\Leftrightarrow C \cdot \lambda_i = k \cdot \mu_i$
 $\therefore \lambda_i = X \cdot \mu_i$

fig2.2 证明

得到特征值与特征向量后，根据特征值排序重新组合特征向量，按照“从粗糙到细致”的思想排序，按照选择的能量百分比保留主成分的个数（PCs）。一般来说，做人脸检测的话较少的 PCs 就足够了，然而想要效果比较良好的人脸重构，还是需要较多的 PCs。此时的矩阵每一列就是一个特征脸。

```

eigvals, eigVects = np.linalg.eig(np.mat(diffTrain.T * diffTrain)) # 205个特征向量
eigSortIndex = np.argsort(-eigvals) # 特征值排序
for i in range(np.shape(FaceMat)[1]):
    if (eigvals[eigSortIndex[i]] / eigvals.sum()).sum() >= energy: # 按energy参数保留
        eigSortIndex = eigSortIndex[:i]
        break
eigen_face = diffTrain * eigVects[:, eigSortIndex] # 得到特征脸(在0.85energy下是10304x47)

return mean_face, eigen_face, diffTrain

```

fig2.3 得到特征矩阵

至此，训练部分结束。可以将上述结果导出到 model 文件里。

```

def export_model(mean_face, eigen_face, diffTrain, filename):
    """
    保存model到目标json文件
    """
    model = {'mean_face':mean_face.__str__(), 'eigen_face':eigen_face.__str__(), 'diffTrain':diffTrain.__str__()}
    json.dump(model, open(filename, 'w'))

```

fig2.4 训练结果导出为 json

3. 识别过程

将输入的图像扁平化后，减去平均脸，得到测试人脸的偏差值图像。用我们之前得到的特征矩阵转置与之相乘，可以得到一个每一项为对应特征脸权重的列向量（维度为 $PCs \times 1$ ）。

检测相似度的时候，用各个训练数据得到的权重与上面算出的测试脸的权重求欧氏距离，来判断最相似的脸。

```

def EigenFaceTesting(img_col, mean_face, eigen_face, diffTrain):
    """
    测试，返回训练集中找到的最相似图像
    """

    diffTest = img_col - mean_face # 测试得到的偏差图像
    weightVec = eigen_face.T * diffTest # 权重 110x1

    res = 0
    resVal = np.inf
    for i in range(np.shape(diffTrain)[1]):
        TrainVec = eigen_face.T * diffTrain[:,i]
        if (np.array(weightVec - TrainVec)**2).sum() < resVal:
            res = i
            resVal = (np.array(weightVec-TrainVec)**2).sum()
    res_img = 'att_faces/s' + str(res//5+1) + '/' + str(res%5+1) + '.pgm'
    return res_img

```

fig3.1 评估最相似人脸

4. 重建过程

重建过程与检测过程思路类似，注意先对每一个特征点除以其欧氏距离（即列向量的 2-范数）。

从平均脸开始重建，逐渐增加主成分个数（PCs）。用测试图像的偏差矩阵的转置乘以对应的特征向量，得到权重。权重反乘这个特征脸，叠加到平均脸上即可得到重建的结果。

```
img = cv2.imread(filename, 0)
img_col = np.array(img, dtype='float64').flatten()
img_col = np.reshape(img_col, (10304,1))
diffTest = img_col - mean_face # 差异图像
# weightVec = eigen_face.T * diffTest
eigen_face = eigen_face / np.linalg.norm(eigen_face, axis=0)
# 重构人脸的时候要做一下这一步!!

for i in range(eigen_face.shape[1]):
    weight = diffTest.T * eigen_face[:,i]
    # print(weight.shape)
    output = output + float(weight) * eigen_face[:,i].reshape(112,92)
```

fig4.1 重建人脸关键代码

四、 实验结果与分析

训练过程得到的结果如下：

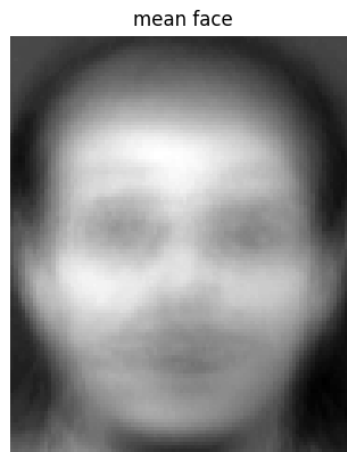


fig5.1 平均脸

eigen faces



eigen faces mixed



fig5.2 10 个特征脸与混合结果

在能量选择 0.8 的情况下的最相似人脸的识别结果正确率还是很高的。

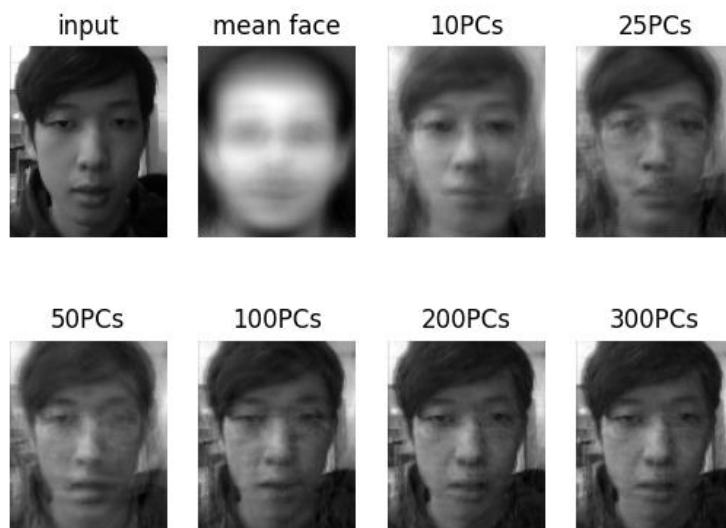
test result



fig5.3 最匹配人脸检测结果

下面是对自己人脸的重建结果：

reconstruct result



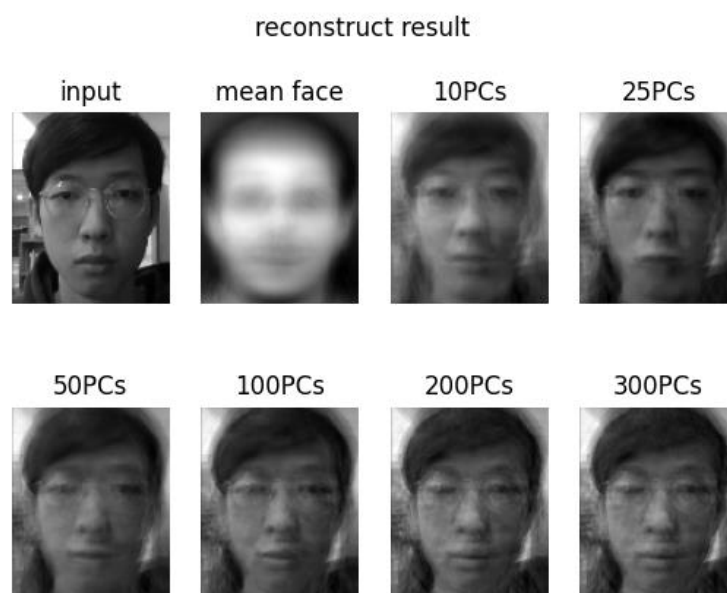


fig5.4 自己的人脸的重建过程（无眼镜、有眼镜）

重建时，我重新导出了一个更大的 model：它使用了每组数据的 9 张图做训练，使用能量百分比为 0.99，生成的特征矩阵有 300 个主成分。在 300PCs 时重建出的结果已经与原图非常接近。

最后，调整能量百分比，每次用一半数据做训练另一半数据做检测，得到 Rank-1 随 PCs 增加的曲线（也就是正确率的曲线图）如下：

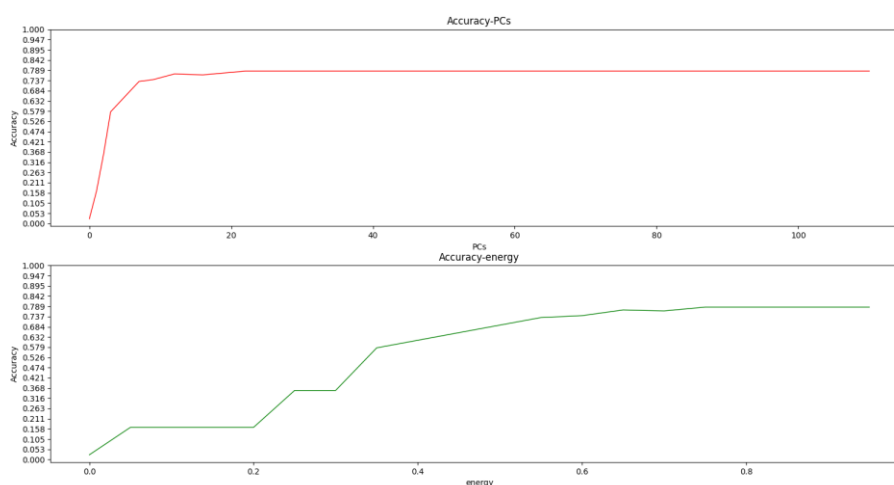


fig5.5 Rank-1 随 PCs 增加或能量百分比增加的曲线图

五、 心得与体会

本次实验采用了 python 编程，代码量比较少，主要是使用了 numpy 的数学处理函数。但是因为 1.样本图像比较大；2.训练数据比较大，所以跑起来需要花大量的时间，而且在导出模型的过程中要花费更长的时间（需要打印很长的特征矩阵），导出的模型数据量也很大。

另外深刻体会到了自己线性代数基础的薄弱，需要好好补课啦！

对 PCA 算法和 eigenface 算法有了亲身实践体会，还是很有趣的。对于空间映射的思路也有了进一步的理解。