HW04 Eigenface

3180101041 杨锐

1.软件开发说明

1.开发环境

- MacOS
- Python == 3.7.6
- Numpy == 1.8.1
- Opencv-python == 4.4.0
- Matplotlib.pyplot == 3.1.2
- Click == 7.0

2.运行方式

```
# 使用40份AT&T人脸+一份自己的人脸中的前5张做训练
python eigenface.py train
# 使用40份AT&T人脸+一份自己的人脸中的后5张做识别测试
python eigenface.py test
# 使用masked_1.jpg进行重构
python eigenface.py reconstruct masked_1.jpg
```

2.算法设计说明

1.PCA思想

EigenFace方法的核心思想是PCA。在多元统计分析中，**主成分分析**（英语：**Principal components analysis**，**PCA**）是一种统计分析、简化数据集的方法。它利用正交变换来对一系列可能相关的变量的观测值进行线性变换，从而投影为一系列线性不相关变量的值，这些不相关变量称为主成分（Principal Components）。具体地，主成分可以看做一个线性方程，其包含一系列线性系数来指示投影方向。PCA对原始数据的正则化或预处理敏感（相对缩放）。

基本思想：

- 将坐标轴中心移到数据的中心，然后旋转坐标轴，使得数据在C1轴上的方差最大，即全部n个数据个体在该方向上的投影最为分散。意味着更多的信息被保留下来。C1成为**第一主成分**。
- C2**第二主成分**：找一个C2，使得C2与C1的协方差（相关系数）为0，以免与C1信息重叠，并且使数据在该方向的方差尽量最大。
- 以此类推，找到第三主成分，第四主成分……第p个主成分。p个随机变量可以有p个主成分。

这样说还不是很清晰，我们通过数学公式来理解。

设我们拥有一个数据集：每一列都是一个n维的字段

$$X = \begin{pmatrix} a_1 & a_2 & \cdots & a_m \end{pmatrix}$$

作:

$$C = \frac{1}{m} X X^T$$

其中:

C是一个对称矩阵，其对角线分别个各个字段的方差，而第i行j列和j行i列元素相同，表示i和j两个字段的协方差。

目标:

将协方差矩阵C对角化，即除对角线外的其它元素化为0，并且在对角线上将元素按大小从上到下排列

因此我们进一步观察原矩阵与基变换后矩阵协方差矩阵的关系:

设原始数据矩阵X对应的协方差矩阵为C，而P是一组基按行组成的矩阵，设Y=PX，则Y为X对P做基变换后的数据。设Y的协方差矩阵为D，则有:

$$D = PCP^T$$

而C是实对称矩阵，则一定能找到:

$$E = \begin{pmatrix} e_1 & e_2 & \cdots & e_n \end{pmatrix}$$

其中:

e_i为C的单位正交特征向量。

那么:

$$E^{\mathsf{T}} C E = \Lambda = \begin{pmatrix} \lambda_1 & & & \\ & \lambda_2 & & \\ & & \ddots & \\ & & & \lambda_n \end{pmatrix}$$

因此我们便得到了数据集X的n维特征基(n PCS)

$$P = E^{\mathsf{T}}$$

设P通过对特征值($eigen\ value$)降序排列，将对应特征向量$eigen\ vector$按行排列，$K = P[0:k]$便是X的k维特征基，即前k个主成分。

换句话说，$Y = KX$成功将数据集X的维度从n降至k。

至此，我们便完成了PCA的原理分析。

2.算法流程

在Eigenface中，PCA的具体应用是对训练集中所有人脸图像的协方差矩阵进行特征值分解，得到对应的特征向量，这些特征向量就是“特征脸”。每个特征向量或者特征脸相当于捕捉或者描述人脸之间的一种变化或者特性。这就意味着每个人脸都可以表示为这些特征脸的线性组合。即我们可以将对每个人脸进行PCA空间变换，将原来$m*n$维的数据映射到k维的空间坐标，相当于每个人脸都有PCA空间对应的一个k维的坐标，那么我们就可以通过两个点之间欧式距离评估两张人脸之间的相似度。

训练

- 将已经进行预处理的人脸图片灰度处理、直方图均衡化。
- 将处理过的图片拉长为1维数组。
- 对所有图片求均值，得到平均脸。
- 将所有图片减去平均脸，得到差值（方便求协方差矩阵）。
- 利用公式$C=\frac{1}{m}XX^T$求出协方差矩阵。
- 求出C的特征值和特征向量，并按特征值降序排列特征向量。
- 根据后续实验要求，最多使用100个PCS，取出前100个保存至本地model.txt
- 取前10个特征向量，拉伸为二维图片，打印并保存。

识别

- 根据输入pcs的数量读取从model.txt中读取对应数量的特征向量
- 将训练库映射到特征空间中（矩阵左乘法）
- 将当前测试图片映射到特征空间中
- 遍历训练库坐标，计算最小欧式距离和对应原始图片地址
- 判断识别是否正确
- 返回第三步，继续识别。
- 计算当前PCS准确率。

重构

- 根据输入pcs的数量读取从model.txt中读取对应数量的特征向量
- 将输入人脸照片映射到特征空间
- 将上一步映射结果再映射回原始空间(左乘特征空间转置矩阵)
- 保存结果，继续下一个pcs
- 将重构结果拼接在一张图，显示并保存

3.算法细节

训练

- 读入图片并灰度处理、直方图均衡化，拉长为1维

```
img_gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
# Normalization
img_norm = cv2.equalizeHist(img_gray)
height, width = img_norm.shape
# 2D img to 1D array
img_array = img_norm.reshape(
    height * width, 1).astype('float64')
```

- 计算平均脸和"差值脸"

```
# 平均脸
avg_face = np.mean(faces, axis=0)
avg_img = avg_face.reshape((height, width))
cv2.imwrite("./train_result/avg_face.jpg", avg_img.astype('uint8'))
cv2.imshow("avgface", avg_img.astype('uint8'))
print("Avg Face finished...")
# 减去平均脸，方便求协方差矩阵
fai_faces = []
for face in faces:
    fai_face = face - avg_face
    fai_faces.append(fai_face)
print("Delta face finished")
```

- 求协方差矩阵

```
# 求Cov矩阵
cov = np.zeros((height*width, height*width))
for delta_face in fai_faces:
    cov += np.transpose(delta_face) * delta_face
cov = cov / len(fai_faces)
```

- 求特征值和特征向量，并按特征值降序排列特征向量

```
# 求特征值和特征向量并降序排列
eigenvalue, eigenvector = np.linalg.eig(cov)
sorted_ev = []
for i in range(height*width):
    sorted_ev.append((eigenvalue[i].real, i))
    sorted_ev.sort(reverse=True, key=lambda e: e[0])
```

- 取出前100个特征向量，保存到本地

```
# 保存前100个特征向量
for i in range(100):
    v.append(eigenvector[:, sorted_ev[i][1]].real)
np.savetxt('./model/model.txt', np.array(v), fmt="%s")
```

- 取出前十个特征向量，还原为人脸并显示

```
for i in range(1, 10 + 1):
    v_img.append(np.array(v[i-1]).reshape(height, width))
    cv2.normalize(v_img[i-1], v_img[i-1], 0, 255, cv2.NORM_MINMAX)
    plt.subplot(2, 5, i)
    plt.imshow(v_img[i-1].astype('uint8'), cmap=plt.cm.gray)
    plt.xticks(())
    plt.yticks(())
plt.savefig("./train_result/eigenfaces.jpg"
plt.show()
```

识别

- 设定pcs，读入待识别图片，记录识别结果

```
# 使用40份AT&T人脸+一份自己的人脸中的后5张做识别测试
dataset_path = './train'
res_list = []
for pcs in [10, 25, 50, 100]:
    result_cnt = 0
    for i in range(1, 42):
        person_path = f'{dataset_path}/s{i}'
        for j in range(6, 11):
            face_path = f'{person_path}/masked_{j}.jpg'
            result_cnt += 1 if EigenFace.mytester(face_path, pcs) else 0
    res = f'PCs: {pcs}\nSuccess: {result_cnt}\nTotal: 205\nrate:
{result_cnt/205}%\n\n'
    print(res)
    res_list.append(res)
with open(f'./test_ans.txt', 'w+') as file:
    file.writelines(res_list)
    file.close()
```

- 读取model

```
# 读取model
A = np.loadtxt('./model/model.txt')
A = A[0:(pcs-1)]
```

- 计算测试图片在特征空间的坐标

```
# 计算测试图片在特征空间的坐标
test_img = cv2.imread(test_path)
test_gray = cv2.cvtColor(test_img, cv2.COLOR_BGR2GRAY)
test_norm = cv2.equalizeHist(test_gray)
height, width = test_norm.shape
test_array = test_norm.reshape((height*width), 1).astype('float64')
test_y = np.matmul(A, test_array)
```

- 遍历训练库，寻找特征空间下欧氏距离最近的图片

```
# 将训练库中的图片映射到特征空间
y = np.matmul(A, img_array)
# 计算欧氏距离最小的图片即最相似
distance = np.linalg.norm(y - test_y)
if distance < min:
    min = distance
    result_path = face_path
```

- 判断是否识别正确

```
# 识别是否正确
(_, _, result_person, _) = result_path.split('/')
(_, _, test_person, test_face) = test_path.split('/')
output_path = f'./test_result_{pcs}/{test_person}/test_{test_face}'
if result_person == test_person:
    test_result = True
test_info = 'Success!' if test_result else 'Failed!'
```

- 显示原图/识别图/叠合图在一张图片并保存到本地
- 返回识别结果

重构

- 根据pcs读取相应的特征向量
- 读取输入人脸和平均脸，拉长为一维，作差

```
img_array = img_norm.reshape(height * width, 1)
avg_face = cv2.imread('./train_result/avg_face.jpg', flags=-1)
avg_array = avg_face.reshape(height * width, 1)
img_float_array = img_array.astype(
    'float64') - avg_array.astype('float64')
```

- 将作差后的结果做特征空间的映射

```
y = np.matmul(A[0:p[i]], img_float_array)
```

- 重构

```
rec_result.append(np.matmul(A[0:p[i]].T, y).astype('float64'))
rec_result[i] = np.array(rec_result[i]).reshape(height, width)
rec_result[i] += avg_face
cv2.normalize(rec_result[i], rec_result[i],0, 255, cv2.NORM_MINMAX)
```

- 显示最终重构结果到一张图片上

实验结果分析

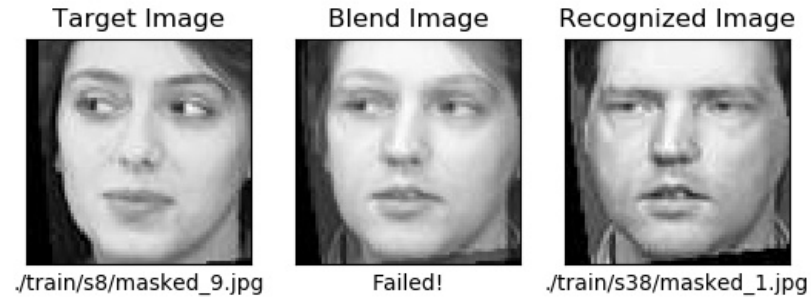
- 训练
  - 平均脸

  - 前十张特征脸

- 识别
  - AT&T

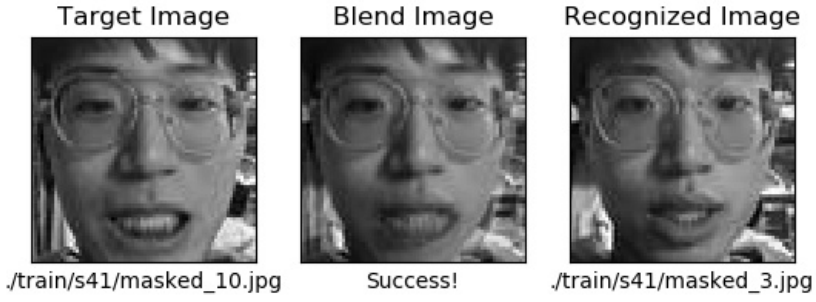
Success1

Success2

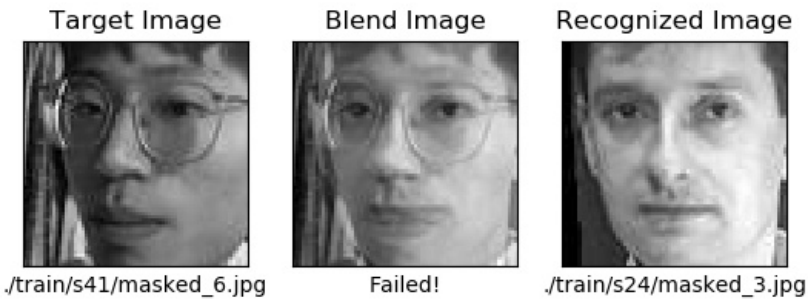


Failed

- Me

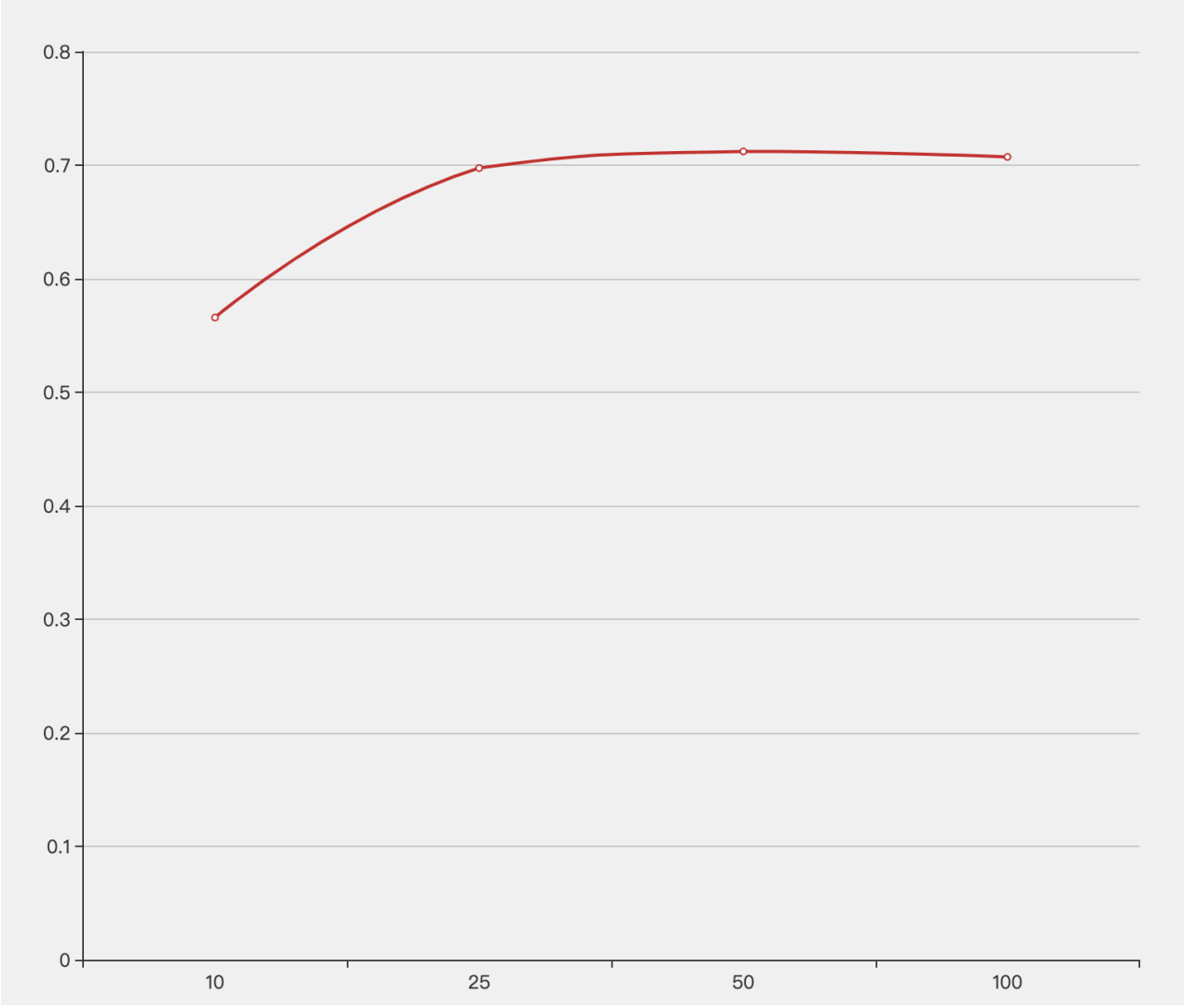


Success

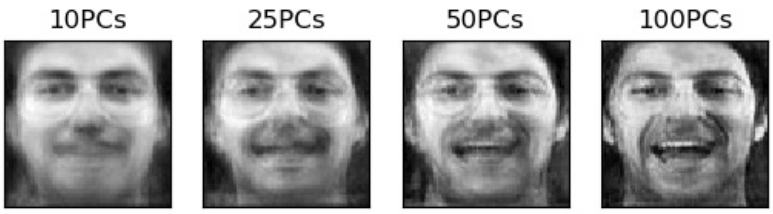


Failed

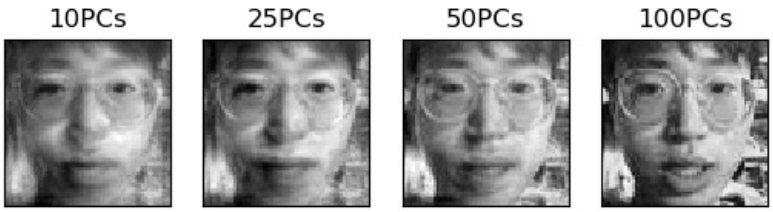
识别成功率随PCS变化曲线



- 重构
  - AT&T



  - Me



编程体会

- 为了理解PCA算法的原理，复习了一些线性代数特别是矩阵相关的知识（当初没学好...），自己一步步把从一个数据集到求主成分空间整个过程推导了一遍，感觉收获很大。
- 在弄懂原理之后，具体实现时也碰到了不少麻烦，比如一开始我没有对图片进行眼睛对齐，这样不同图片的尺寸是不一样的。导致了在识别时和重构时，平均脸无法和原始人脸重叠时对齐，效果较差。因此我重新对图片进行了手工选择眼睛，并根据眼睛的位置裁剪所有图片到相同尺寸。这样重

构的效果明显更好。

- 但对图片进行裁剪，使得某些人脸图片丢失了部分信息，这也导致了识别率只能维持在70%左右。
- 另一个问题则是数据类型，由于所有图片像素经过灰度处理之后都为*uint8[0:255]*,而在识别和重构时都需要进行矩阵运算，显示这个精度是不够。我们需要将图片先转换为*float64*，矩阵运算，然后使用`cv2.normalize()`进行归一化，最后再转换为*uint8*保存为灰度图片。
- 感觉这次实验比前几次的难度都要大，主要在于PCA算法原理的理解和代码中对图片、矩阵处理过程中的细节。总的来说，这次实验收获很大！

个人照片