Assignment 4

姓名：田原

学号：3180101981

开发软件说明

- 主要开发语言
  - Python 3.8
- 主要开发环境
  - Pycharm
- 编译系统环境
  - Windows 10 (64-bit)

算法具体步骤及实现要点

预处理

预处理的思路：根据人脸两只眼睛的中心位置，求出仿射变换矩阵，进行旋转缩放，再执行直方图均衡化操作。

- 感谢宋天泽同学在钉钉群提供的眼睛中心位置标注数据！这里要注意部分txt文件保留了多行数据，每次读取前两行即可。
- 由于提供的数据集已经把脸的位置框出来，这里不需要再切除脸部区域，根据眼睛位置做一定旋转即可。

事实上，这部分工作在与同学交流后，我发现，这部分处理对我们的最终效果其实基本没有提升，**甚至还有所下降**……有部分同学认为不需要执行把眼睛对齐的操作，处理成灰度图就已经有不错的效果。但是这部分代码还挺有趣的，所以还是写在这里吧：

具体实现代码如下：

`load_eyes_pos`函数，把眼睛的位置从文件中读取出来：

```
def load_eyes_pos(pos_src):
    fr = open(pos_src)  # txt数据文件
    eyes_pos = np.zeros((2, 2))
    index = 0
    for line in fr.readlines():
        line = line.strip()
        from_line = line.split(',')
        eyes_pos[index, :] = from_line[0:2]
        index += 1
        if index == 2:
            break
    return eyes_pos # 返回眼睛位置
```

`transform`函数，对图片执行旋转变换：

```
def transform(gray_img, eyes_pos):
    x1 = eyes_pos[0][1]
    x2 = eyes_pos[1][1]
    y1 = eyes_pos[0][0]
    y2 = eyes_pos[1][0]
    center = ((x1 + x2) / 2, (y1 + y2) / 2)  # 眼睛中心
    angle = math.atan((y2 - y1) / (x2 - x1)) * 180.0 / math.pi  # 旋转角度
    trans_mat = cv2.getRotationMatrix2D(center, angle, 1.0)  # 仿射变换矩阵
    trans_mat[0][2] = trans_mat[0][2] + 37.0 - center[0]
    trans_mat[1][2] = trans_mat[1][2] + 30.0 - center[1]
    rows, cols = gray_img.shape[:2]
    trans_img = cv2.warpAffine(gray_img, trans_mat, (math.floor(cols * 4 / 5),
math.floor(rows * 4 / 5)))  # 根据变换矩阵旋转缩放
    trans_img = cv2.equalizeHist(trans_img)  # 均衡化
    return trans_img
```

`read_data`函数，把处理后的图片写到对应路径：

```
def read_data(img_src, write_src):
    for i in range(10):
        path = img_src + str(i + 1) + ".pgm"
        path1 = img_src + str(i + 1) + ".txt"
        image = cv2.imread(path)
        eyes_pos = load_eyes_pos(path1)
        image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
        image = transform(image, eyes_pos)  # 变换
        # cv2.imshow("1", image)  # 执行时不需要，所以注释了，取消注释可以看到图片快速切
换，
        # cv2.waitKey(100)  #       每张图片的眼睛位置都是基本一致的。
        is_exists = os.path.exists(write_src)
        if not is_exists:
            os.makedirs(write_src)
        cv2.imwrite(write_src + str(i + 1) + ".pgm", image)  # 写到对应文件夹
```

执行部分：

```
path = "./att_faces_with_eyes/"
path1 = "./data/"
for i in range(40):
    read_src = path + "s" + str(i+1) + "/"
    write_src = path1 + "s" + str(i+1) + "/"
    read_data(read_src, write_src)
```

执行完毕后，处理好的图片就保存在了相应的data文件夹。

训练

训练部分的主要思路如下：

- 求平均脸mean face记为 M 并保存。
- 计算每张图片和 M 的差值矩阵 Ω_i，求协方差矩阵 Σ：$\Sigma = \frac{1}{K}\Sigma_{i=1}^{K}\Omega_i\Omega_i^T$。（MN*MN）
- 计算协方差矩阵的特征向量，归一化得到特征脸和训练集所有图片对应的特征脸权重。

此处要注意，直接计算特征向量的话计算量非常大，如果训练图像的数量 n 小于图像维数（本实验中是height * width），那么起作用的特征向量其实只有n个，所以求特征向量只需要一个N*N的矩阵，比M*M的计算量小很多。

具体代码如下：

```
# the train process
def mytrain(feature_num, model_file_name, images):
    image = np.array(images)  # train pictures together
    n = image.shape[0]  # number of pictures
    height = image.shape[1]  # height
    width = image.shape[2]  # width
    capa = height * width
    # calculate mean face 计算平均脸
    image_arr = []
    for i in range(n):
        image_arr.append(image[i].flatten())
    mean_arr = np.mean(np.array(image_arr).T, axis=1).astype(np.uint8)
    mean_image = (np.mean(np.array(image_arr).T,
axis=1).astype(np.uint8)).reshape((height, width))
    cv2.imwrite("mean_face.png", mean_image)  # write mean face
    # calculate covariance matrix and get the eigenvectors 求特征向量
    normalized = []
    for i in range(n):
        normalized.append(image_arr[i] - mean_arr)
    X = np.array(normalized).T
    if n > capa:  # 如果图片数量大于维数
        C = np.dot(X, X.T)  # C是M*M的矩阵
        [eigenvalues, eigenvectors] = np.linalg.eig(C)
    else:  # 否则
        C = np.dot(X.T, X)  # C是N*N的矩阵
        [eigenvalues, eigenvectors] = np.linalg.eig(C)
        eigenvectors = np.dot(X, eigenvectors)  # 再乘回去得到原来的特征向量
    idx = np.argsort(-eigenvalues)
    eigenvectors = eigenvectors[:, idx]
    eigenvectors = eigenvectors[:, 0:feature_num].copy()
    # normalize
    for i in range(feature_num):
```

```
        eigenvectors[:, i] = (eigenvectors[:, i] - eigenvectors[:, i].min()) /
(eigenvectors[:, i].max() - eigenvectors[:, i].min())
    # get 10 eigenface image
    img = np.zeros((height * 1, width * 10))
    for i in range(feature_num):
        tmp = np.asarray(eigenvectors[:, i]).reshape((height, width))
        tmp = 255 * tmp
        x = i % 10
        img[0 : height, x * width:(x + 1) * width] = tmp
    cv2.imwrite("eigenface.png", img)   # 保存特征脸图片
    # save model
    np.save(model_file_name, eigenvectors)
    weight = []
    for i in range(n):
        weight.append(np.matmul(normalized[i], eigenvectors))
    return mean_arr, np.asarray(weight)
```

具体的代码结果在结果展示部分。

识别

识别部分的主要思路如下：

- 考虑一张新的人脸，我们可以用特征脸对其进行标示。计算出每个特征脸对应的权重，构成一个向量。
- 对于训练集内的每个人脸，都对应有一个特征脸的权重，构成一个向量。
- 计算两个向量的欧氏距离，最短距离所对应的图片为识别结果。

这部分思路比较简单，具体代码如下：

```
# identify function
# 对img进行图像识别，选出最接近的一张人脸
def mytest(img, model_file_name, mean_arr, train_weight):
    # img: image to identify
    # model_file_name: train model file name
    # mean_arr: mean image
    # train_weight: train vector
    height = img.shape[0]  # height
    width = img.shape[1]  # width
    eigenvectors = np.load(model_file_name) # 得到特征向量
    # img - mean
    img = np.asarray(img).flatten() - mean_arr
    # 特征脸对人脸的表示
    weight = np.matmul(img.T, eigenvectors)
    min_dis = np.sum((train_weight[0] - weight) ** 2)
    index = 0
    for i in range(train_weight.shape[0]):  # 遍历训练集的所有特征脸权重
        dist = np.sum((train_weight[i] - weight) ** 2)  # 求欧氏距离
        dist = dist ** 0.5
        if (dist < min_dis):  # find the closest face
            min_dis = dist
            index = i # 图片编号
    # calculate the dir path
    x = math.floor(index / 5) + 1
    y = index % 5 + 1
    min_img = cv2.imread(path + "s" + str(x) + "/" + str(y) + ".pgm")
    min_img = cv2.cvtColor(min_img, cv2.COLOR_BGR2GRAY)
```

```
    show_img = np.hstack([ori_img, min_img])  # 把原图和识别图放在一起展示
    # show the result
    cv2.imshow("show identify result", show_img)  # 把原图和识别图放在一起展示
    cv2.waitKey(0)
    return x, y
```

重构

将model文件装载进来后，对输入的人脸图像进行变换到特征脸空间，然后再用变换后的结果重构回原来人脸图像。

```
# reconstruct process
def myreconstruct(img, model_file_name, pc_num):
    height = img.shape[0]
    width = img.shape[1]
    eigenvectors = np.load(model_file_name)  # load特征向量
    f = (np.matmul(eigenvectors, np.matmul(eigenvectors.T,
(img.flatten()).T))).reshape((height, width))  # 重构图片
    f = 255 * (f - f.min()) / (f.max() - f.min())
    cv2.imwrite("reconstruct" + str(pc_num) + ".jpg", f)
```

执行中的代码如下：

```
# reconstruct
pc_list = [10, 25, 50, 100]
for i in range(len(pc_list)):
    mean_arr, train_weight = mytrain(pc_list[i], "model.npy", images,
norm_method=1)
    img = cv2.imread("trans.jpg")  # 已经经过transform处理的我自己的脸
    img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    myreconstruct(img, "model.npy", pc_list[i])  # 进行重构
```

结果展示及分析

平均人脸与特征脸

平均人脸如下：

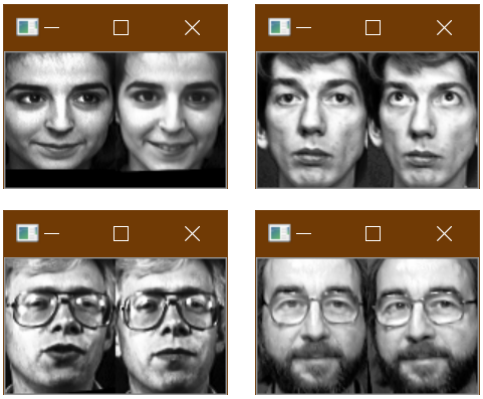
由于代码对齐了眼睛部分，能看出来眉眼鼻区域比较清晰。由于样本基本都是欧洲人的脸，所以平均脸看起来具有眼睛深邃、鼻子较高的特征。

特征脸如下：

一开始我看到这个结果以为是自己写错了……和同学交流发现如果有降维操作基本都是这样的结果。和PPT上的结果还是不太一样的，推测PPT用了更复杂或者精巧的写法？

部分识别结果

实验主体代码在`main.py`文件中，用每个人前五张人脸作为训练集，测试识别时用后面的5张图片测试。展示4张识别结果如下：



人脸重构结果

首先是对我的人脸的处理，处理成黑白再对齐眼睛：

->

然后执行重构，结果如下：

10PCs

25PCs

50PCs

100PCs

识别率的变化曲线

下图展示随着PC增加，识别率的变化曲线。基本能达到75%的识别率。