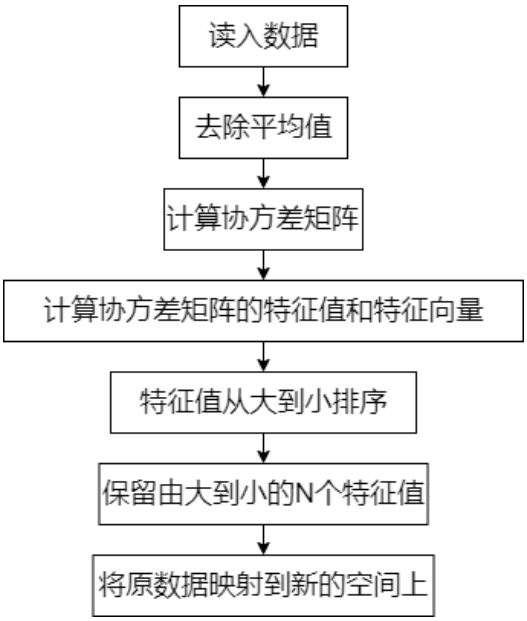
单类数据降维实践大作业

PCA 人脸识别实验

（1） 降维

- 算法流程图



- 源代码

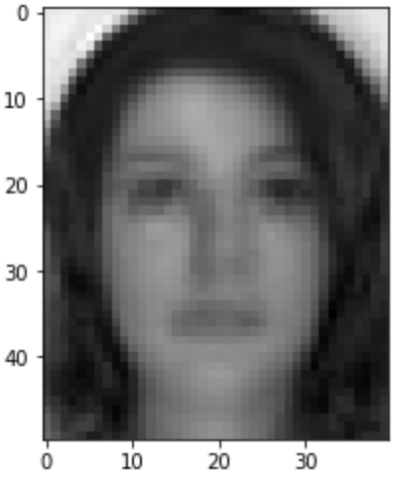
```
#定义PCA算法
def PCA(data,r):
    '''
    input
    data:降维数据,维度:[dims,image_nums]
    r:降维维度

    return
    final_data:降维后结果,维度:[image_nums,r]
    data_mean:平均值,维度:[1,image_nums]
    V_r:特征值向量,维度:[dims,r]
    '''
    data=np.float32(np.mat(data))#将输入数据转换为numpy中的array形式
    rows,cols=np.shape(data)#得到行列值
    data_mean=np.mean(data,0)#对列求平均值
    A=data-np.tile(data_mean,(rows,1))#去除平均值
    C=A*A.T #计算得到协方差矩阵
    D,V=np.linalg.eig(C)#求协方差矩阵的特征值和特征向量
    V_r=V[:,0:r]#按列取前r个特征向量
    V_r=A.T*V_r#小矩阵特征向量向大矩阵特征向量过渡
    for i in range(r):
        V_r[:,i]=V_r[:,i]/np.linalg.norm(V_r[:,i])#特征向量归一化
    final_data=A*V_r#将原数据映射到新的空间上
    return final_data,data_mean,V_r
```

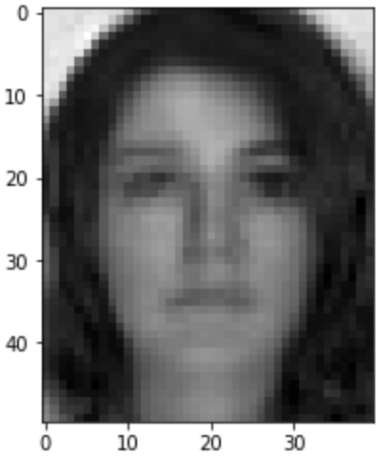
- 实验

A. 对比原图像和降维后重建的图像

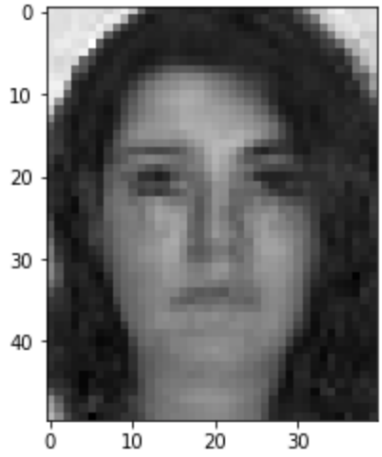
原图像



重建图像（k=50）



重建图像（k=100）



重建图像（k=200）

重建图像（k=500）

重建图像（k=1000）

B. 计算重建误差

降维的重建误差用投影距离来表示，公式如下所示：

$$\frac{\frac{1}{m}\sum_{i=1}^{m}\|x^{(i)}-x_{approx}^{(i)}\|^2}{\frac{1}{m}\sum_{i=1}^{m}\|x^{(i)}\|^2}$$

其中 m 表示特征值的个数，分子表示原始点与投影点之间的距离之和，而误差越小，说明降维后的数据越能完整表示降维前的数据。如果这个误差小于 0.01，说明降维后的数据能保留 99%的信息。

代码如下：

```
#计算重建误差
def loss_clc(re_mat,img_mat):
    #维度:[dims,image_nums]
    re_mat = np.array(re_mat)
    img_mat = np.array(img_mat)
    t1 = img_mat - re_mat
    t1 = np.multiply(t1,t1)
    t2 = np.sum(t1,axis=0)
    t3 = np.mean(t2)#计算分子
    #精度问题处理
    s1 = img_mat/10
    s1 = np.multiply(s1,s1)
    s2 = np.sum(s1,axis=0)
    s3 = np.mean(s2)#计算分母
    loss = t3/(s3*100)
    print("重建误差为: %.5f,保留了 %d %%的信息"%(loss,(1-loss)*100))
    return loss
```

K=50 时，重建结果为：

```
pca_ed(50)
```

```
当前降维维度k=50
重建误差为: 0.02210,保留了 97 %的信息
```

K=100 时，重建结果为：

```
pca_ed(100)
```

```
当前降维维度k=100
重建误差为: 0.01382,保留了 98 %的信息
```

K=200 时，重建结果为：

```
pca_ed(200)
```

```
当前降维维度k=200
重建误差为: 0.00761,保留了 99 %的信息
```

K=500 时，重建结果为：

```
pca_ed(500)
```

```
当前降维维度k=500
重建误差为: 0.00213,保留了 99 %的信息
```

K=1000 时，重建结果为：

```
pca_ed(1000)
```

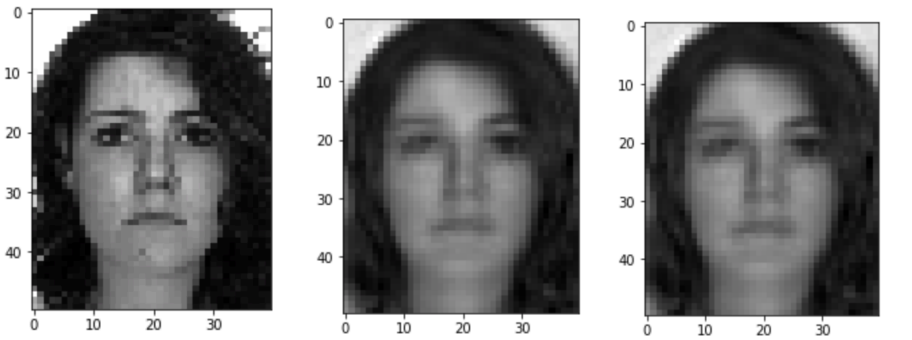
```
当前降维维度k=1000
重建误差为: 0.00022,保留了 99 %的信息
```

C. 与 sklearn 中的 PCA 方法对比

Sklearn 库中已经有成熟的包，因此直接调用其中的 PCA 方法与实验中自己编写的程序进行对比。

实现代码：

```
#使用sklearn的方法进行实验
k=100
pca = skl_PCA(k)#from sklearn.decomposition import PCA as skl_PCA
newX = pca.fit_transform(train_imgMat)#输入数据进行降维，返回降维结果
re_newX = pca.inverse_transform(newX)#重建方法
print("当前降维维度k=",k)
loss = loss_clc(re_newX,train_imgMat)
show_pca = re_newX[14].reshape(50,40)
plt.imshow(show_pca,cmap='gray')
plt.show()
```



原图像　　实验程序重建图像（k=200）　　sklearn 重建图像（k=200）

K=50 时，sklearn 的重建结果为：

```
当前降维维度k= 50
重建误差为: 0.02211,保留了 97 %的信息
```

K=100 时，sklearn 的重建结果为：

```
当前降维维度k= 100
重建误差为: 0.01384,保留了 98 %的信息
```

K=200 时，sklearn 的重建结果为：

```
当前降维维度k= 200
重建误差为: 0.00767,保留了 99 %的信息
```

与之相对应的实验 PCA 的结果：

```
pca_ed(50)
```

```
当前降维维度k=50
重建误差为: 0.02210,保留了 97 %的信息
```

```
pca_ed(100)
```

```
当前降维维度k=100
重建误差为: 0.01382,保留了 98 %的信息
```

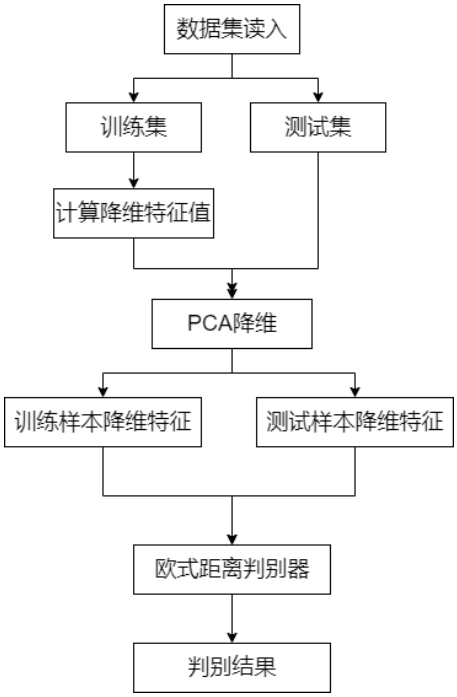
```
pca_ed(200)
```

```
当前降维维度k=200
重建误差为: 0.00761,保留了 99 %的信息
```

经过对比后可以发现，调用 sklearn 中的 PCA 的实验结果与自己编写程序算法的误差基本一致，算法验证了本实验算法的有效性。

（2） 人脸识别

- 算法流程图



- 源代码

1. 创建数据集

对同一张人脸，选择 12 张图片作为训练集，2 张图片作为测试集。

```
path="/Users/ren/Desktop/AR"
train_x=[]
train_y=[]
test_x=[]
test_y=[]
pca = PCA(n_components=35)
for file in os.listdir(path):
    if file[-3:]!="bmp":
            continue
    no = int(file[4:6])#取同一张人脸的编号
    if(no>=3 and no<=14):#取后12张人脸作为训练集
        train_tempImg = cv2.imread(path+'/'+file,0)#读入图片
        train_tempImg = np.array(train_tempImg)
        train_tempImg = train_tempImg.reshape(2000)#将二维图片向量转换为以为特征向量
        train_x.append(train_tempImg)
        train_y.append(int(file[:3]))#取不同人脸编号作为label值
    else:#取前2张人脸作为测试集
        test_tempImg = cv2.imread(path+'/'+file,0)
        test_tempImg = np.array(test_tempImg)
        test_tempImg = test_tempImg.reshape(2000)
        test_x.append(test_tempImg)
        test_y.append(int(file[:3]))
train_x = np.array(train_x)#将列表形式转换为数组形式
train_y = np.array(train_y)
test_x = np.array(test_x)
test_y = np.array(test_y)
train_x.reshape(-1,2000)#确保返回形状为[image_nums,size]
test_x.reshape(-1,2000)
```

2. PCA 降维

```python
#定义PCA算法
def PCA(data,r):
    '''
    input
    data:降维数据,维度:[dims,image_nums]
    r:降维维度

    return
    final_data:降维后结果,维度:[image_nums,r]
    data_mean:平均值,维度:[1,image_nums]
    V_r:特征值向量,维度:[dims,r]
    '''
    data=np.float32(np.mat(data))#将输入数据转换为numpy中的array形式
    rows,cols=np.shape(data)#得到行列值
    data_mean=np.mean(data,0)#对列求平均值
    A=data-np.tile(data_mean,(rows,1))#去除平均值
    C=A*A.T #计算得到协方差矩阵
    D,V=np.linalg.eig(C)#求协方差矩阵的特征值和特征向量
    V_r=V[:,0:r]#按列取前r个特征向量
    V_r=A.T*V_r#小矩阵特征向量向大矩阵特征向量过渡
    for i in range(r):
        V_r[:,i]=V_r[:,i]/np.linalg.norm(V_r[:,i])#特征向量归一化
    final_data=A*V_r#将原数据映射到新的空间上
    return final_data,data_mean,V_r
```

3. 欧式距离判别器

```python
#欧式距离判别器
def ed(num_test,num_train,data_test_new,data_train_new,train_label,test_label):
    true_num = 0#判别正确的值
    for i in range(num_test):#遍历测试集中的图片
        testFace = data_test_new[i,:]#取当前图片的特征值
        diffMat = data_train_new - np.tile(testFace,(num_train,1))
        sqDiffMat = diffMat**2#计算训练数据与测试脸之间的欧式距离
        sqDistances = sqDiffMat.sum(axis=1)#按行求和
        sortedDistIndicies = sqDistances.argsort()#对向量从小到大排序，使用的是索引值,得到一个向量
        indexMin = sortedDistIndicies[0]#距离最近的索引
        if train_label[indexMin] == test_label[i]:#判别是否预测正确
            true_num += 1
    return true_num
```

4. 误差计算

```python
#计算重建误差
def loss_clc(re_mat,img_mat):
    #维度:[dims,image_nums]
    re_mat = np.array(re_mat)
    img_mat = np.array(img_mat)
    t1 = img_mat - re_mat
    t1 = np.multiply(t1,t1)
    t2 = np.sum(t1,axis=0)
    t3 = np.mean(t2)#计算分子
    #此处因为254*254会超出numpy.multiply的范围，做了除10的操作，后面求loss时乘了回来
    s1 = img_mat/10
    s1 = np.multiply(s1,s1)
    s2 = np.sum(s1,axis=0)
    s3 = np.mean(s2)#计算分母
    loss = t3/(s3*100)
    print("重建误差为：%.5f,保留了 %d %%的信息"%(loss,(1-loss)*100))
    return loss
```

5. 主程序

```
#使用PCA降维+欧氏距离判别
def pca_ed(k=200):
    print("当前降维维度k=%d"%(k))
    data_train_new,data_mean,V_r=PCA(train_x,k)#用训练集计算特征值，返回训练集降维结果
    num_train = data_train_new.shape[0]#训练脸总数
    num_test = test_x.shape[0]#测试脸总数
    temp_face = test_x - np.tile(data_mean,(num_test,1))#测试集去中心值
    data_test_new = temp_face*V_r #得到测试脸在特征向量下的数据
    data_test_new = np.array(data_test_new) # 转换为数组形式
    data_train_new = np.array(data_train_new)
    true_num = ed(num_test,num_train,data_test_new,data_train_new,train_y,test_y)#计算判别正确的数目
    re_train = re_creat(data_train_new,V_r,data_mean)#降维后重建
    old_mat = train_x.T
    new_mat = re_train.T
    loss = loss_clc(new_mat,old_mat)#计算重构误差
    print("当前测试集大小为：%d,判别正确数量为：%d"%(num_test,true_num))
    print("当前判别准确率为：%.5f"%(true_num/num_test))
    #绘图时的输出
    #print("当前降维维度k=%d,当前判别准确率为：%.5f"%(k,(true_num/num_test)))
    return true_num/num_test
```

6. 结果展示

```
list_x=[]
list_y=[]
#三个梯队进行测试
for i in range(1,50,2):
    list_x.append(i)
    list_y.append(pca_ed(i))
for i in range(52,400,10):
    list_x.append(i)
    list_y.append(pca_ed(i))
for i in range(500,1000,50):
    list_x.append(i)
    list_y.append(pca_ed(i))
```

```
plt.plot(list_x, list_y, label="pca_acc", linewidth=1.5)
plt.xlabel('D')#维度
plt.ylabel('acc')#准确率
plt.legend()
plt.show()
```

- 实验结果

分别对 k=2，k=5，k=15，k=50，k=100，k=200 的情况进行实验，实验结果如下：

K=2 时，判别结果为：

```
pca_ed(2)
```

```
当前降维维度k=2
重建误差为：0.09637,保留了 90 %的信息
当前测试集大小为：240,判别正确数量为：46
当前判别准确率为：0.19167
```

K=5 时，判别结果为：

```
pca_ed(5)
```

```
当前降维维度k=5
重建误差为：0.06390,保留了 93 %的信息
当前测试集大小为：240,判别正确数量为：179
当前判别准确率为：0.74583
```

K=15 时，判别结果为：

```
pca_ed(15)
```

```
当前降维维度k=15
重建误差为：0.04069,保留了 95 %的信息
当前测试集大小为：240,判别正确数量为：218
当前判别准确率为：0.90833
```

K=50 时，判别结果为：

```
pca_ed(50)
```

```
当前降维维度k=50
重建误差为：0.02210,保留了 97 %的信息
当前测试集大小为：240,判别正确数量为：227
当前判别准确率为：0.94583
```

K=100 时，判别结果为：

```
pca_ed(100)
```

```
当前降维维度k=100
重建误差为：0.01382,保留了 98 %的信息
当前测试集大小为：240,判别正确数量为：227
当前判别准确率为：0.94583
```

K=200 时，判别结果为：

```
pca_ed(200)
```

```
当前降维维度k=200
重建误差为：0.00761,保留了 99 %的信息
当前测试集大小为：240,判别正确数量为：230
当前判别准确率为：0.95833
```

- 绘制结果曲线

  A. 实验代码

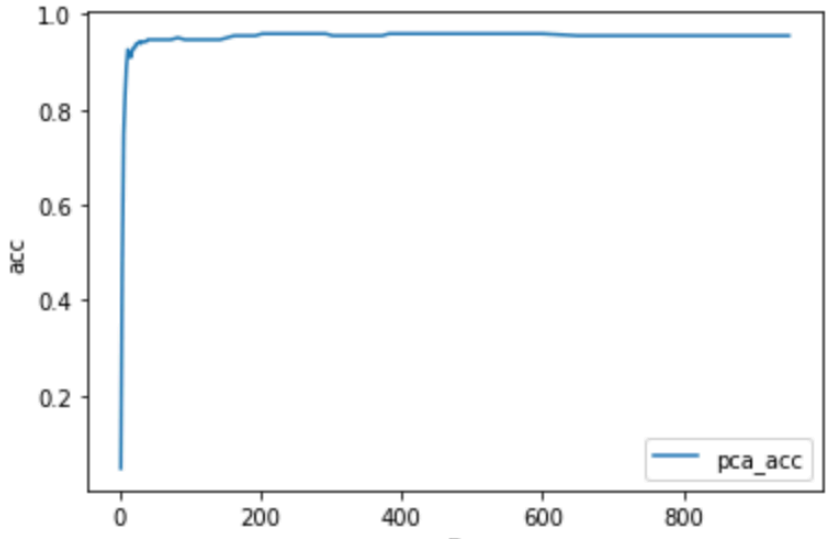
```
list_x=[]
list_y=[]
#三个梯队进行测试
for i in range(1,50,2):
    list_x.append(i)
    list_y.append(pca_ed(i))
for i in range(52,400,10):
    list_x.append(i)
    list_y.append(pca_ed(i))
for i in range(500,1000,50):
    list_x.append(i)
    list_y.append(pca_ed(i))
```

```
plt.plot(list_x, list_y, label="pca_acc", linewidth=1.5)
plt.xlabel('D')#维度
plt.ylabel('acc')#准确率
plt.legend()
plt.show()
```

B. 实验数据

```
当前降维维度k=1,当前判别准确率为: 0.04583
当前降维维度k=3,当前判别准确率为: 0.46667
当前降维维度k=5,当前判别准确率为: 0.74583
当前降维维度k=7,当前判别准确率为: 0.82917
当前降维维度k=9,当前判别准确率为: 0.89167
当前降维维度k=11,当前判别准确率为: 0.92500
当前降维维度k=13,当前判别准确率为: 0.91667
当前降维维度k=15,当前判别准确率为: 0.90833
当前降维维度k=17,当前判别准确率为: 0.92083
当前降维维度k=19,当前判别准确率为: 0.92917
当前降维维度k=21,当前判别准确率为: 0.92917
当前降维维度k=23,当前判别准确率为: 0.93750
当前降维维度k=25,当前判别准确率为: 0.93750
当前降维维度k=27,当前判别准确率为: 0.94167
当前降维维度k=29,当前判别准确率为: 0.93750
当前降维维度k=31,当前判别准确率为: 0.94167
当前降维维度k=33,当前判别准确率为: 0.94167
当前降维维度k=35,当前判别准确率为: 0.94167
当前降维维度k=37,当前判别准确率为: 0.94167
当前降维维度k=39,当前判别准确率为: 0.94583
当前降维维度k=41,当前判别准确率为: 0.94583
当前降维维度k=43,当前判别准确率为: 0.94583
当前降维维度k=45,当前判别准确率为: 0.94583
当前降维维度k=47,当前判别准确率为: 0.94583
当前降维维度k=49,当前判别准确率为: 0.94583
当前降维维度k=52,当前判别准确率为: 0.94583
当前降维维度k=62,当前判别准确率为: 0.94583
当前降维维度k=72,当前判别准确率为: 0.94583
当前降维维度k=82,当前判别准确率为: 0.95000
当前降维维度k=92,当前判别准确率为: 0.94583
当前降维维度k=102,当前判别准确率为: 0.94583
当前降维维度k=112,当前判别准确率为: 0.94583
当前降维维度k=122,当前判别准确率为: 0.94583
当前降维维度k=132,当前判别准确率为: 0.94583
当前降维维度k=142,当前判别准确率为: 0.94583
当前降维维度k=152,当前判别准确率为: 0.95000
当前降维维度k=162,当前判别准确率为: 0.95417
当前降维维度k=172,当前判别准确率为: 0.95417
当前降维维度k=182,当前判别准确率为: 0.95417
当前降维维度k=192,当前判别准确率为: 0.95417
当前降维维度k=202,当前判别准确率为: 0.95833
当前降维维度k=212,当前判别准确率为: 0.95833
当前降维维度k=222,当前判别准确率为: 0.95833
当前降维维度k=232,当前判别准确率为: 0.95833
当前降维维度k=242,当前判别准确率为: 0.95833
当前降维维度k=252,当前判别准确率为: 0.95833
当前降维维度k=262,当前判别准确率为: 0.95833
当前降维维度k=272,当前判别准确率为: 0.95833
当前降维维度k=282,当前判别准确率为: 0.95833
当前降维维度k=292,当前判别准确率为: 0.95833
当前降维维度k=302,当前判别准确率为: 0.95417
当前降维维度k=312,当前判别准确率为: 0.95417
当前降维维度k=322,当前判别准确率为: 0.95417
当前降维维度k=332,当前判别准确率为: 0.95417
当前降维维度k=342,当前判别准确率为: 0.95417
当前降维维度k=352,当前判别准确率为: 0.95417
当前降维维度k=362,当前判别准确率为: 0.95417
当前降维维度k=372,当前判别准确率为: 0.95417
当前降维维度k=382,当前判别准确率为: 0.95833
当前降维维度k=392,当前判别准确率为: 0.95833
当前降维维度k=500,当前判别准确率为: 0.95833
当前降维维度k=550,当前判别准确率为: 0.95833
当前降维维度k=600,当前判别准确率为: 0.95833
当前降维维度k=650,当前判别准确率为: 0.95417
当前降维维度k=700,当前判别准确率为: 0.95417
当前降维维度k=750,当前判别准确率为: 0.95417
当前降维维度k=800,当前判别准确率为: 0.95417
当前降维维度k=850,当前判别准确率为: 0.95417
当前降维维度k=900,当前判别准确率为: 0.95417
当前降维维度k=950,当前判别准确率为: 0.95417
```

C. 实验折线图



- **实验结果分析**

从实验结果可以看出，随着 k 取值的增大，保留原始信息比例的增加，判别的准确率随之不断提升，对于该实验集，k=25 时，准确率基本满足要求，之后随着 k 的上升，判别率上涨幅度不是特别明显，甚至到收敛时 acc 比 k=25 时仅提高了 2%的点，但是基本收敛时 k 大概增大了 10 倍。因此在实际应用中，可根据计算资源的紧缺程度和对精度的需求选取合适的 k 值。

- **实验心得**

之前学习 PCA 只是感觉是种理论上的东西，甚至对于其压缩后还原后表示怀疑，但是经过自己实验后发现确实有效，不得不感慨 PCA 的强大之处。通过本次实验锻炼了自己的动手能力，也彻底的学会了 PCA 在图片上的应用，为日后在计算机领域的发展打好了基础，挖掘了兴趣。