哈尔滨工业大学计算机科学与技术学院

实验报告

课程名称： 机器学习

课程类型： 考查课

实验题目： PCA 模型实验

学号：

姓名：

一、实验目的

实现一个 PCA 模型，能够对给定数据进行降维（即找到其中的主成分）。

二、实验要求及实验环境

①实验要求：

（1）首先人工生成一些数据（如三维数据），让它们主要分布在低维空间中，如首先让某个维度的方差远小于其它唯独，然后对这些数据旋转。生成这些数据后，用你的 PCA 方法进行主成分提取。

（2）找一个人脸数据（小点样本量），用你实现 PCA 方法对该数据降维，找出一些主成分，然后用这些主成分对每一副人脸图像进行重建，比较一些它们与原图像有多大差别（用信噪比衡量）。

②实验环境：

Windows10; python 3.9.12; Visual Studio Code; Anaconda.

三、设计思想（本程序中的用到的主要算法及数据结构）

①主成分分析 PCA：

PCA(主成分分析，Principal Component Analysis)是最常用的一种降维方法。PCA 的主要思想是将 D 维特征通过一组投影向量映射到 K 维上，这 K 维是全新的正交特征，称之为主成分，采用主成分作为数据的代表，有效地降低了数据维度，且保留了最多的信息。关于 PCA 的推导有两种方式：最大投影方差和最小投影距离。

最大投影方差：样本点在这个超平面上的投影尽可能分开

最小投影距离：样本点到这个超平面的距离都足够近

推导过程：

开始 PCA 之前需要对数据进行预处理，即对数据中心化。设数据集 $X = \{x_1, x_2, \cdots\cdots, x_n\}$，其中 $x_i = \{x_{i1}, x_{i2}, \cdots\cdots, x_{id}\}$，即 X 是一个 n*d 的矩阵。则此数据集的中心向量（均值向量）为：

$$\mu = \frac{1}{n}\sum_{i=1}^{n} x_i$$

对数据集每个样本均进行操作：$x_i = x_i - \mu$. 就得到了中心化后的数据，此时有

$$\sum_{i=1}^{n} x_i = 0$$

中心化可以给后面的计算带来极大的便利，因为中心化之后的常规线性变换就是绕原点的旋转变化，也就是坐标变换。此时，协方差为:

$$S = \frac{1}{n}\sum_{i=1}^{n} x_i x_i^T = \frac{1}{n} X^T X$$

设使用的投影坐标系的一组标准正交基为 $U_{k*d} = \{u_1, u_2, \cdots\cdots, u_k\}$，$k<d$，$u_i = \{u_{i1}, u_{i2}, \cdots\cdots, u_{id}\}$. 故有 $UU^T = 1$. 使用这组基变换中心化矩阵 X，得降维压缩后的矩阵 $Y_{n*k} = XU$. 重建得到 $X = YU = XU^TU$.

对于任意一个样本 x_i，在新的坐标系中的投影为 $y_i = x_iU^T$，在新

坐标系中的投影方差为 $y_i^T y_i = Ux_i^T x_i U^T$。要使所有的样本的投影方差和最大，即：

$$\arg\max_U \ tr(UX^T XU^T) \qquad s.t.\ UU^T = 1$$

求解：在 u_i 方向投影后的方差：

$$\frac{1}{n}\sum_{i=1}^{n}\{u_1^T x_i - u_1^T \mu\}^2 = \frac{1}{n}(Xu_1^T)^T(Xu_1^T) = \frac{1}{n}u_1 X^T X u_1^T = u_1 S u_1^T$$

因为 u_1 是投影方向，且已经假设它是单位向量 $u_i u_i^T = 1$，用拉格朗日乘子法最大化目标函数：

$$L(u_1) = u_1^T S u_1 + \lambda_1(1 - u_1^T u_1)$$

对 u_1 求导，令导数等于 0，解得 $Su_1 = \lambda_1 u_1$，显然，u_1 和 λ_1 是一组对应的 S 的特征向量和特征值, 所以有 $u_1^T S u_1 = \lambda_1$.，结合在 u_1 方向投影后的方差式，可得求得最大化方差，等价于求最大的特征值。

要将 d 维的数据降维到 k 维，只需计算前 k 个最大的特征值，将其对应的特征向量（d*1 的）转为行向量（1*d 的）组合成特征向量矩阵 U_{k*d}，则降维压缩后的矩阵为 $Y = X U^T$.

```
# 将训练集 trainset 从 D 维降到 k 维，返回重构之后的数据矩阵（N * D）
def PCA(trainset, k):
    trainset = trainset.T
    dimension = trainset.shape[0]
    mean = np.mean(trainset, axis = 1)
    norm_set = np.zeros(trainset.shape)
    for i in range(dimension):
        # 零均值化后得到 norm_set (D*N)
        norm_set[i] = trainset[i] - mean[i]
    # 求出协方差矩阵
    covMatrix = np.dot(norm_set, norm_set.T)
    # 对协方差矩阵 covMatrix(D*D)求特征值和特征向量
```

```
    # eigenVectors 为 D*D 矩阵，每一列对应一个特征向量
    eigenValues, eigenVectors = np.linalg.eig(covMatrix)
    # 特征值按升序排序，并返回排序后的索引
    eigValIndex = np.argsort(eigenValues)
    # 取前 k 个特征值对应的特征向量(D*k)
    KEigenVector = eigenVectors[:, eigValIndex[:-(k + 1):-1]]
    # numpy.linalg.eig 计算的特征向量在计算的时候是以复数的形式运算的
    # 算法在收敛时，虚部可能还没有完全收敛到 0，可以对其保留实部
    KEigenVector = np.real(KEigenVector)
    # 计算降维后的数据(K*N)
    reduce_tmp_set = np.dot(KEigenVector.T, norm_set)
    # 重构之后的数据
    result_set = np.zeros(trainset.shape)
    for i in range(dimension):
        # 重构的数据 = 重构后的各方向标量大小 * 单位方向向量 + 该方向均值
        result_set[i] = np.dot(KEigenVector[i], reduce_tmp_set) + mean[i]
    return result_set.T
```

②峰值信噪比 PSNR：

PSNR 全称是 Peak signal-to-noise ratio，是一种客观的量化评价视频质量的方法。把原始参考视频与失真视频在每一个对应帧中的每一个对应像素之间进行比较。准确的讲，这种方法得到的并不是真正的视频质量，而是失真视频相对于原始视频的相似程度或保真程度。

在图像和视频传输领域中，压缩是常见的。压缩的过程是去除冗余信息，解压缩的过程是重构冗余信息。在去除与重构过程中，会导致部分信息的丢失或出现错误。峰值信噪比用来衡量对压缩后的图像或视频进行重建的质量，它常简单地通过均方差（MSE）进行定义。两个 m*n 灰度图 I 和 K，如果一个为另一个的噪声近似（原图的重构），那他们的均方差定义为：

$$MSE = \frac{1}{m\,n}\sum_{i=0}^{m-1}\sum_{j=0}^{n-1}[I(i,j) - K(i,j)]^2.$$

峰值信噪比定义为：

$$\mathrm{PSNR} = 10\log_{10}\frac{\mathrm{MAX}^2}{\mathrm{MSE}}$$

一般来说，图像与影像压缩中典型的峰值讯噪比值在 30dB 到 50dB 之间，愈高愈好。

PSNR 接近 50dB ，代表压缩后的图像仅有些许非常小的误差。

PSNR 大于 30dB ，人眼很难察觉压缩后和原始影像的差异。

PSNR 介于 20dB 到 30dB 之间，人眼就可以察觉出图像的差异。

PSNR 介于 10dB 到 20dB 之间，人眼还是可以用肉眼看出这个图像原始的结构，且直观上会判断两张图像不存在很大的差异。

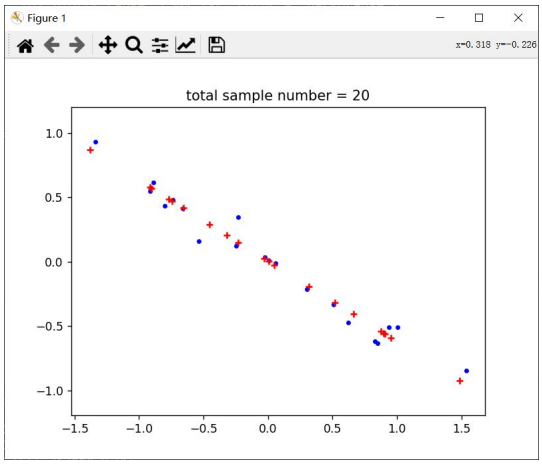
PSNR 低于 10dB，人类很难用肉眼去判断两个图像是否为相同，一个图像是否为另一个图像的压缩结果。

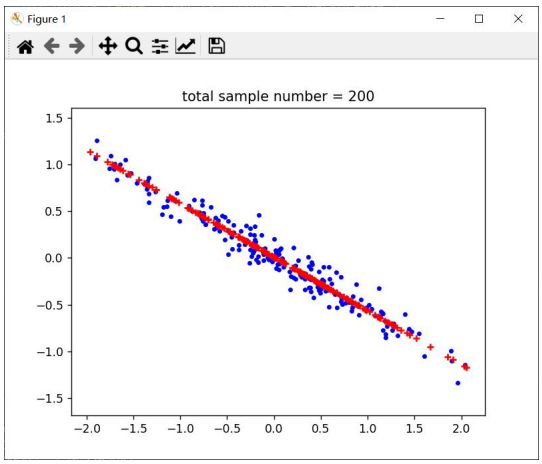
```
# 计算前 num 张图片的信噪比、平均信噪比
def compute_PSNR(trainset, result_set, num):
    MSE = np.zeros(num)
    PSNR = np.zeros(num)
    for i in range(num):
        MSE[i] += np.mean(((trainset[i, :, :].reshape(-1,3) - result_set[i, :, :].reshape(-1,3)))
** 2 )
        PSNR[i] = 20 * np.log10(255 / np.sqrt(MSE[i]))
    return PSNR, np.mean(PSNR)
```

四、实验结果与分析

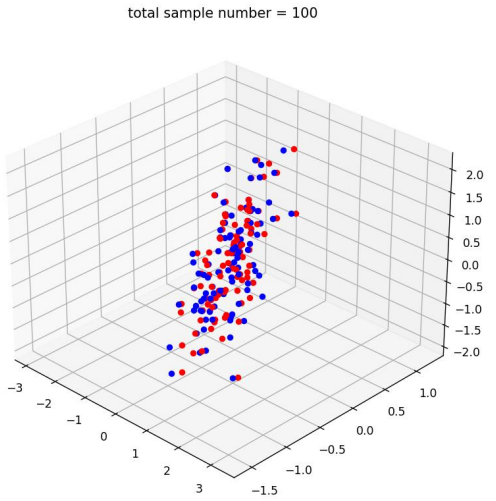
①人工生成数据：

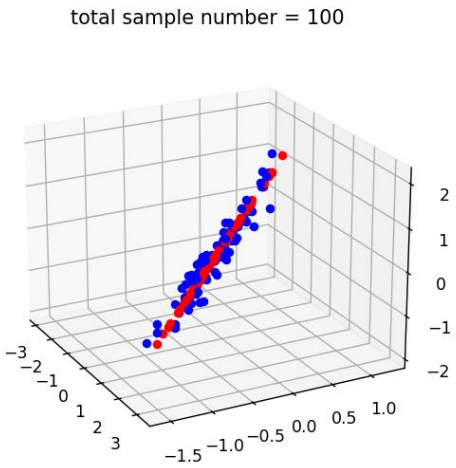
1.二维数据降维成一维数据



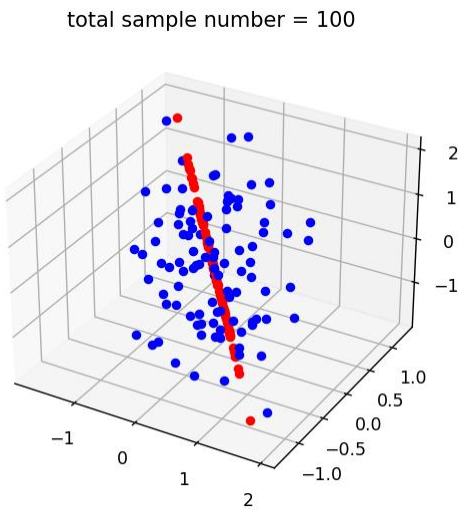


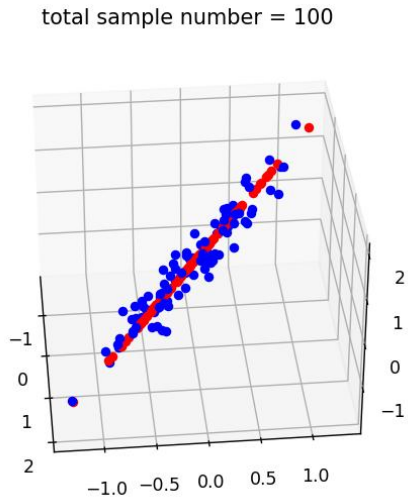
2. 三维数据降维成二维数据





3. 三维数据降维成一维数据





从中可以看出，如果数据本身集中在高维空间的某一些维度

上，通过 PCA 降维能很大程度上降低维度、压缩存储空间，并且信息的损失也比较小。但是如果降维程度过大，如把三维空间的二维分布强行降维到一维，会损失很多信息，将原本较大的方差去除。

②人脸数据降维：

1. 有背景、差异大、彩色人脸数据集：

数据集简介：

该数据集来自 CeleA，是香港中文大学的开放数据，包含 10177 个名人身份的 202599 张图片，并且都做好了特征标记，背景和人脸相对杂乱。由于算力有限，本实验从中抽取了前 2000 张照片进行训练，并每一张图片压缩成 44*54=2376 大小后进行计算，部分照片如下所示：

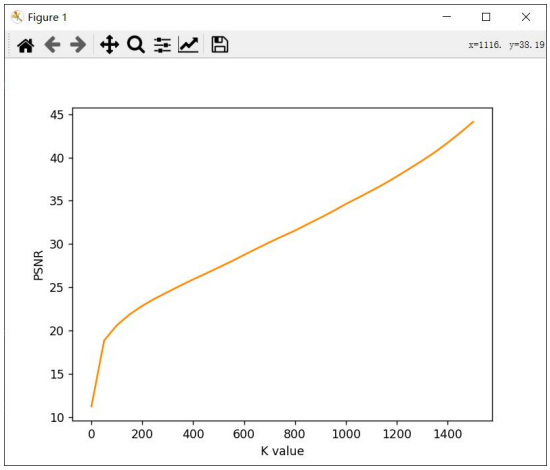
K = 20 时：

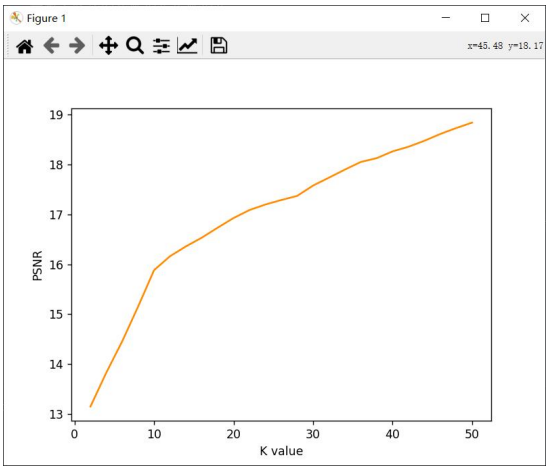
K = 60 时：

K = 500 时：

从中可以看出，当 K=20 时照片非常模糊，人像只有极少数发色等特征彼此不同且符合原图，各个照片人脸极其相近。但是当 K=60 时，人脸的特征已经稍微接近原图了，一些背景、头发位置等信息正确地呈现了出来，能够看出原图的对应关系。K=500 时，图像表现很接近原图，信息还原较多，只是能看出略显模糊。

信噪比随着 K 值的变化：





原始图像为 w*h=2376 维，从峰值信噪比可以看出，大约当 K>60 以后，PSNR 已经大于 20，人眼就可以察觉出图像的差异，但能对应原图。当大约 K>650 后，PSNR>30，人眼很难察觉压缩后和原始影像的差异。当 K 逐渐增长，PSNR 也大约呈线性增长。因此，即便人脸图像样本量较大、背景等差异性较大、通过 PCA 降维，也能够在减少约 3/4 样本储存空间的前提下，让人眼很难察觉对图像的信息量的减小。

2. 无背景、差异小、黑白人脸数据集：

JAFFE :The Japanses Female Facial Expression Database 即日本女性面部表情数据库，该数据库共有 213 张表情图片，由 10 个女性的 7 种表情图片组成。由于算力有限，本实验从中抽取了前 100 张照片进行训练，并每一张图片压缩成 64*64=4096 大小后进行计算，如下图所示：

K = 5 时：

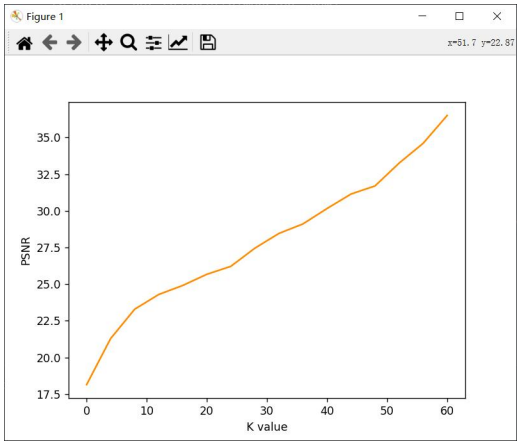
K = 10 时：

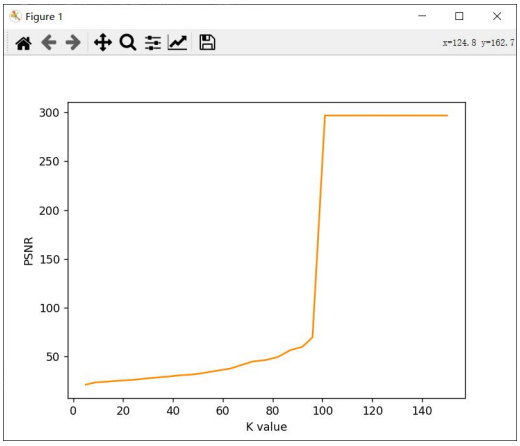
K = 20 时：

从中可以看出，虽然 K=5 时照片已经能够清晰地辨认出不同人

像，但是各个人表情都几乎看不出变化。但是当 K=20 时，表情的变化已经接近原图了，能够看出大部分表情变化。K=10 的图像表现介于 5-20 之间。

信噪比随着 K 值的变化：





原始图像为 w*h=4096 维，但是从峰值信噪比可以看出，当 K>3 以后，PSNR 已经大于 20，人眼就可以轻微察觉出图像的差异。当 K>40 后，PSNR>30，人眼很难察觉压缩后和原始影像的差异。当 K 接近 100 时，PSNR 陡然增大，与原图几乎没有差异。因此，通过 PCA 降维，能够在大量减少相似人脸样本储存空间的前提下，对图像的信息量的减小很轻微。

五、结论

1. PCA 降低了训练数据的维度的同时保留了主要信息，舍弃“次要成分”； 但是对于测试集而言，被舍弃的也许正好是重要的信息，也就是说 PCA 可能会加剧过拟合；

2. PCA 算法中舍弃了 d-k 个最小的特征值对应的特征向量，一定会导致低维空间与高维空间不同，但是通过这种方式有效提高了样本的采

样密度；并且由于较小特征值对应的往往与噪声相关，通过 PCA 在一定程度上起到了降噪的效果。

3. PCA 用于图片的降维可以极大地缓解存储压力，训练集种的图片越是彼此相似、信息量单一，降维储存的效果越好，这样能够避免“维度灾难”。

六、参考文献

[1] Guenin B， Knemann J， Tunel L . A Gentle Introduction to Optimization. 2018.

[2] https://zh.wikipedia.org/wiki/

七、附录：源代码（带注释）

```
import math
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
import cv2
import os


# 对样本点 trainset 绕原点进行二维顺时针旋转 angle 弧度
def rotate_2D(trainset, angle = 0):
    x = trainset[:, 0]
    y = trainset[:, 1]
    Rotate_trainset = np.zeros((trainset.shape))
    Rotate_trainset[:, 0] = x * math.cos(angle) + y * math.sin(angle)
    Rotate_trainset[:, 1] = y * math.cos(angle) - x * math.sin(angle)
    return Rotate_trainset
```

```
# 对样本点 trainset 绕 axis 轴三维旋转 angle 弧度
def rotate_3D(trainset, axis, angle = 0):
    if axis == 'x':
```

```
        rotate_matrix = [[1, 0, 0], [0, np.cos(angle), -np.sin(angle)], [0, np.sin(angle),
np.cos(angle)]]
    elif axis == 'y':
        rotate_matrix = [[np.cos(angle), 0, np.sin(angle)], [0, 1, 0], [-np.sin(angle), 0,
np.cos(angle)]]
    elif axis == 'z':
        rotate_matrix = [[np.cos(angle), -np.sin(angle), 0], [np.sin(angle), np.cos(angle), 0],
[0, 0, 1]]
    else:
        assert False
    return np.dot(rotate_matrix, trainset.T).T
```

```
# 准备：其中 dimension 为样本原始的维度，为了可视化 demension 只取 2 或 3,num 为样本点数量
def get_trainset(dimension, num):
    if dimension == 2:
        mean = np.array([0, 0])
        covXY = np.array([[1, 0], [0, 0.01]])
        trainset = np.random.multivariate_normal(mean, covXY, size = num)
        trainset = rotate_2D(trainset, math.pi / 6)
        return trainset

    elif dimension == 3:
        mean = np.array([0, 0, 0])
        cov = [[0.01, 0, 0], [0, 1, 0], [0, 0, 1]]
        trainset = np.random.multivariate_normal(mean, cov, size = num)
        trainset = rotate_3D(trainset, 'z', math.pi / 3)
        trainset = rotate_3D(trainset, 'y', math.pi / 4)
        return trainset



# 绘制训练集散点图
def draw(trainset, result_set, color1, color2):
    if (trainset.shape[1] == 2):
        for [x,y] in trainset:
            plt.scatter(x, y, color = color1, marker = '.')
            plt.axis('equal')
        for [x,y] in result_set:
            plt.scatter(x, y, color = color2, marker = '+')
            plt.axis('equal')

    elif (trainset.shape[1] == 3):
        fig = plt.figure()
        ax = fig.add_subplot(projection='3d')
```

```
        for [x,y,z] in trainset:
            ax.scatter(x, y, z, s = 20, c = color1, depthshade=True)
        for [x,y,z] in result_set:
            ax.scatter(x, y, z, s = 20, c = color2, depthshade=True)
    else:
        assert False
    plt.title('total sample number = %d' %(trainset.shape[0]))
    plt.show()
```

```
# 将训练集 trainset 从 D 维降到 k 维，返回重构之后的数据矩阵（N * D）
def PCA(trainset, k):
    trainset = trainset.T
    dimension = trainset.shape[0]
    mean = np.mean(trainset, axis = 1)
    norm_set = np.zeros(trainset.shape)
    for i in range(dimension):
        # 零均值化后得到 norm_set (D*N)
        norm_set[i] = trainset[i] - mean[i]
    # 求出协方差矩阵
    covMatrix = np.dot(norm_set, norm_set.T)
    # 对协方差矩阵 covMatrix(D*D)求特征值和特征向量
    # eigenVectors 为 D*D 矩阵，每一列对应一个特征向量
    eigenValues, eigenVectors = np.linalg.eig(covMatrix)
    # 特征值按升序排序，并返回排序后的索引
    eigValIndex = np.argsort(eigenValues)
    # 取前 k 个特征值对应的特征向量(D*k)
    KEigenVector = eigenVectors[:, eigValIndex[:-(k + 1):-1]]
    # numpy.linalg.eig 计算的特征向量在计算的时候是以复数的形式运算的
    # 算法在收敛时，虚部可能还没有完全收敛到 0，可以对其保留实部
    KEigenVector = np.real(KEigenVector)
    # 计算降维后的数据(K*N)
    reduce_tmp_set = np.dot(KEigenVector.T, norm_set)
    # 重构之后的数据
    result_set = np.zeros(trainset.shape)
    for i in range(dimension):
        # 重构的数据 = 重构后的各方向标量大小 * 单位方向向量 + 该方向均值
        result_set[i] = np.dot(KEigenVector[i], reduce_tmp_set) + mean[i]
    return result_set.T
```

```
# 逐一读取 image 文件夹中的图片，并将 RGB 像素值存储到 trainset(16, w * h, 3)中
def read_file_to_trainset(path):
    trainset = []
    img_folder = path
```

```
    img_list = [os.path.join(nm) for nm in os.listdir(img_folder)]
    count = 0
    for i in img_list:
        if count >= 100:
            break
        now_path = path
        now_path = os.path.join(now_path, i)
        img = cv2.imread(now_path)
        # 为了能够快速运行得到结果，对原始图片进行压缩
        size = np.array([img.shape[0]/4, img.shape[1]/4]).astype(int)
        img = cv2.resize(img, size)
        img_reshape = img.reshape((img.shape[0] * img.shape[1], img.shape[2]))
        trainset.append(img_reshape)
        count+=1
    return np.array(trainset), img.shape
```

```
# 显示 trainset 中前 100 张照片
def show_some_figure(trainset, img_shape):
    trainset = trainset[:100]
    fig, ax=plt.subplots(nrows=10, ncols=10, figsize=(10,10))
    for i in range(100):
        img = trainset[i].reshape(img_shape[0], img_shape[1], img_shape[2])
        #imshow 默认信息格式为 BGR，读取的是 RGB 需要转换
        b, g, r = cv2.split(img)
        img_new = cv2.merge([r,g,b])
        ax[i//10, i%10].imshow(img_new)
        ax[i//10, i%10].set_xticks([])
        ax[i//10, i%10].set_yticks([])
    plt.show()
```

```
# 计算前 num 张图片的信噪比、平均信噪比
def compute_PSNR(trainset, result_set, num):
    MSE = np.zeros(num)
    PSNR = np.zeros(num)
    for i in range(num):
        MSE[i] += np.mean(((trainset[i, :, :].reshape(-1,3) - result_set[i, :, :].reshape(-1,3)))
** 2 )
        PSNR[i] = 20 * np.log10(255 / np.sqrt(MSE[i]))
    return PSNR, np.mean(PSNR)
```

```
# 将 trainset 中图片的 R/G/B 值通过 PCA 分别降维到 K 维，返回值格式与 trainset 相同
def RGB_PAC(trainset, K):
```

```
    img_new = np.zeros(trainset.shape)
    for i in range(3):
        img_new[:, :, i] = PCA(trainset[:, :, i], K)
    return img_new
```

```
# 将 trainset 降维到分别降维到 K 维，并计算对应前 num 张图片的平均 PSNR
# 返回对应 K 的 PSNR 列表、并绘制折线图
def get_different_PSNR(trainset, num, K_range):
    PSNR_lst = []
    for K in K_range:
        img_new = RGB_PAC(trainset, K)
        PSNR, mean = compute_PSNR(trainset, img_new, num)
        PSNR_lst.append(mean)

    plt.plot(K_range, PSNR_lst, color = 'darkorange')
    plt.xlabel('K value')
    plt.ylabel('PSNR')
    plt.show()
    return PSNR_lst



if __name__ == '__main__':
    # self_made_data 变量指示：训练时使用自己编造的数据 or 人脸的图片数据
    self_made_data = False

    # 随机生成二维高斯分布的点作为数据集
    if self_made_data == True:
        dimension = 3
        K = 1   # 降维到 k 维
        num = 100
        trainset = get_trainset(dimension, num)
        print (trainset)
        draw(trainset, PCA(trainset, K), 'blue', 'red')

    # 下载人脸的图片数据作为数据集
    if self_made_data == False:
        trainset, img_shape = read_file_to_trainset('3image')
        show_some_figure(trainset[:100], img_shape)
        print (img_shape)
        K = 5
        img_new = RGB_PAC(trainset, K)
        # 因为 PCA 后 RGB 值都是浮点数，除以 255 是为了令 imshow 按照浮点数读取
        show_some_figure(np.clip(img_new/255, 0, 1), img_shape)
```

```
    get_different_PSNR(trainset, 10, np.linspace(0,100,21).astype(int))
```