PCA 图像重建实验报告

姓名：张韫译萱　　学号：08023214

一、 实验题目

图 1: yalefaces 数据集示例图像

1 实验任务

本次实验旨在将 PCA 算法应用于 yalefaces 数据集中图像的降维与重建。

1. 实现 PCA 算法，对 yalefaces 数据集中的面部图像进行降维与重建；
2. 将原始数据的数值范围归一化到 $[0, 1]$ 区间，主成分数量 r 的取值范围为 $r = 5, 10, 15, \ldots, 200$（步长为 5）；
3. 对图像添加不同水平的高斯噪声，噪声标准差 $\sigma \in \{0.0001, 0.001, 0.01, 0.05, 0.1, 0.5, 1\}$；
4. 采用峰值信噪比（Peak Signal-to-Noise Ratio，PSNR）作为图像重建质量的衡量准则。

2 实验要求

1. 选择 C++/Python 实现。

2. 代码以文本形式粘贴在附录相应位置，注释准确，能成功运行。

二、 实验结果

1 PSNR 随主成分数量 r 的变化曲线

我对 7 种不同噪声水平下的 PCA 重建效果进行了实验，绘制了 PSNR 随主成分数量 r 变化的曲线图，如图2所示。

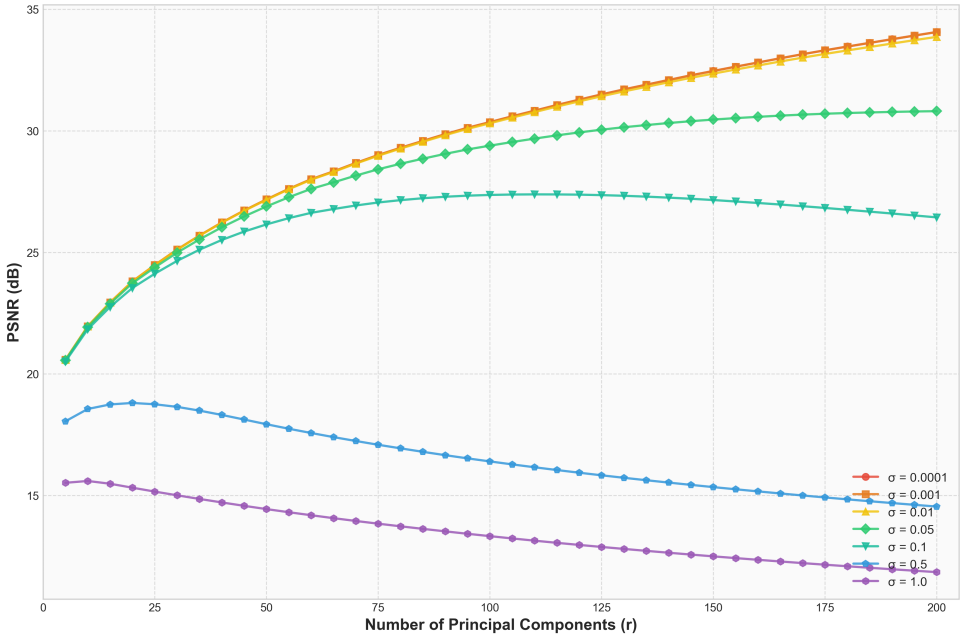

图 2: 不同噪声水平下 PSNR 随主成分数量 r 的变化曲线

2 图像重建效果可视化

我们选取 $r = 20$、$\sigma = 0.05$ 的参数组合，对图像重建前后的效果进行可视化展示，如图3所示。

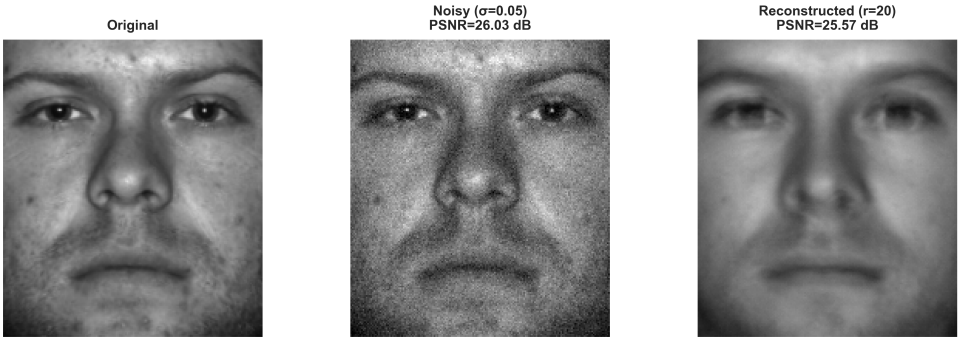

图 3: $r = 20$，$\sigma = 0.05$ 时的图像重建效果对比

三、 实验分析

1 算法原理

主成分分析（Principal Component Analysis，PCA）是一种经典的线性降维方法，其核心思想是通过正交变换将原始数据投影到一组线性无关的新坐标系中，使得投影后的数据方差最大化。

（1）数学原理

设原始数据矩阵为 $\mathbf{X} \in \mathbf{R}^{n\times d}$，其中 n 为样本数量，d 为特征维度。PCA 的目标是找到一组正交基 $\{\mathbf{w}_1, \mathbf{w}_2, \ldots, \mathbf{w}_r\}$，使得数据在这些方向上的投影方差最大。

首先对数据进行中心化处理：

$$\bar{\mathbf{X}} = \mathbf{X} - \mathbf{1}_n \boldsymbol{\mu}^T \tag{1}$$

其中 $\boldsymbol{\mu} = \frac{1}{n}\sum_{i=1}^{n} \mathbf{x}_i$ 为数据均值向量。

计算协方差矩阵：

$$\mathbf{C} = \frac{1}{n-1}\bar{\mathbf{X}}^T\bar{\mathbf{X}} \tag{2}$$

对协方差矩阵进行特征值分解：

$$\mathbf{C}\mathbf{w}_i = \lambda_i \mathbf{w}_i \tag{3}$$

其中 $\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_d$ 为特征值，$\mathbf{w}_i$ 为对应的特征向量（主成分方向）。

（2）SVD 分解实现

实际计算中使用了奇异值分解（SVD）来实现 PCA，以提高数值稳定性（这是计算的第二种方法，和求解协方差矩阵，计算特征向量的方法本质没有区别）：

$$\mathbf{X}\bar{\mathbf{X}}^{\mathbf{T}} = \mathbf{U}\ \mathbf{V}^T \tag{4}$$

其中 $\mathbf{U_x}$ 的列向量即为主成分方向， 对角线上的奇异值与特征值满足 $\sigma_i^2 = (n-1)\lambda_i$。

（3）降维与重建

选取前 r 个主成分构成投影矩阵 $\mathbf{U} = [\mathbf{u}_1, \ldots, \mathbf{u}_r] \in \mathbf{R}^{d\times r}$，降维变换为：

$$\mathbf{Y} = \mathbf{U}^{\mathbf{T}}\mathbf{X} \tag{5}$$

重建过程为降维的逆操作：

$$\mathbf{X} = \mathbf{U}\mathbf{Y} = \mathbf{U}\mathbf{U}^{\mathbf{T}}\bar{\mathbf{X}} \tag{6}$$

（4）PSNR 评价指标

峰值信噪比（PSNR）用于衡量重建图像与原始图像的相似度，定义为：

$$\mathrm{PSNR} = 10 \cdot \log_{10}\left(\frac{\mathrm{MAX}^2}{\mathrm{MSE}}\right) \tag{7}$$

其中 MAX 为像素最大值（归一化后为 1），MSE 为均方误差。PSNR 值越高，表示重建质量越好。

2 实验结果分析

通过对实验结果的深入分析，可以得出以下结论：

1. 从 PSNR 曲线可以观察到，不同噪声水平下存在不同的最优主成分数量 r^*。当噪声较小（$\sigma \leq 0.01$）时，随着 r 的增加，PSNR 持续上升，表明保留更多主成分能够更好地还原图像细节。然而，当噪声较大（$\sigma \geq 0.1$）时，PSNR 在达到峰值后反而下降，这是因为过多的主成分会将噪声信息也纳入重建过程。

2. PCA 具有隐式的降噪能力。由于噪声通常分布在较小的主成分方向上，通过截断这些主成分，PCA 能够有效滤除部分噪声。实验表明，对于中等噪声水平（$\sigma = 0.05$），选取适当的 r 值可以在保留图像主要特征的同时抑制噪声。

3. 主成分数量 r 的选择需要在信息保留与噪声抑制之间取得平衡。r 过小会导致图像细节丢失，r 过大则会引入噪声。实验结果表明，对于含噪图像重建任务，应根据噪声水平自适应地选择 r 值。

4. PSNR 作为客观质量评价指标，能够定量反映重建图像与原始图像之间的差异。从可视化结果来看，PSNR 值的变化与人眼视觉感知基本一致，验证了该指标的有效性。

四、 实验总结

通过本次实验，我对 PCA 算法的理解从理论层面深入到了实践应用。通过实现 PCA 并应用于图像重建任务，我体会到了主成分分析在处理含噪数据时的优雅性，特别是通过截断较小主成分来实现隐式降噪的机制。实验也让我认识到，在实际应用中需要根据具体问题灵活选择主成分数量，以在信息保留与噪声抑制之间找到最佳平衡。

五、 代码

```python
import numpy as np
import matplotlib.pyplot as plt
from PIL import Image
import os
from glob import glob

plt.style.use('ggplot')
np.random.seed(42)


def load_yalefaces(folder_path):  # 加载数据集
    # 尝试读取tif格式，没有就读gif
    image_files = glob(os.path.join(folder_path, '*.tif'))
    if len(image_files) == 0:
        image_files = glob(os.path.join(folder_path, '*.gif'))
    if len(image_files) == 0:
        raise FileNotFoundError(f"在 {folder_path} 中未找到图像文件")

    images = []
    for img_path in sorted(image_files):
```

```
            img = Image.open(img_path).convert('L')  # 转灰度图
            images.append(np.array(img, dtype=np.float64))

    images = np.array(images)
    print(f"加载了 {len(images)} 张图像，尺寸 {images[0].shape}")
    return images, images[0].shape


def normalize_images(images):  # 归一化到[0,1]
    return images / 255.0


def add_gaussian_noise(images, sigma):  # 添加高斯噪声
    noise = np.random.normal(0, sigma, images.shape)
    return np.clip(images + noise, 0, 1)


def pca_fit(X, n_components):  # PCA拟合
    # 中心化
    mean = np.mean(X, axis=0)
    X_centered = X - mean
    # SVD分解，取前n个主成分
    U, S, Vt = np.linalg.svd(X_centered, full_matrices=False)
    return mean, Vt[:n_components]


def pca_transform(X, mean, components):  # 投影到主成分空间
    return np.dot(X - mean, components.T)


def pca_inverse_transform(X_transformed, mean, components):  # 逆变换重建
    return np.dot(X_transformed, components) + mean


def pca_reconstruct(X, n_components):  # PCA降维重建
    mean, components = pca_fit(X, n_components)
    X_transformed = pca_transform(X, mean, components)
    return pca_inverse_transform(X_transformed, mean, components)


def calculate_psnr(original, reconstructed):  # 计算PSNR
    # PSNR = 10 * log10(MAX^2 / MSE)，这里MAX=1
    mse = np.mean((original - reconstructed) ** 2)
    if mse == 0:
        return float('inf')
    return 10 * np.log10(1.0 / mse)


```

```python
def run_experiment(images, r_values, sigma_values):  # 跑实验
    n_samples = images.shape[0]
    n_features = images.shape[1] * images.shape[2]
    X_original = images.reshape(n_samples, n_features)  # 展平成向量

    results = {}
    for sigma in sigma_values:
        print(f"  = {sigma}")
        # 加噪声
        noisy_images = add_gaussian_noise(images, sigma)
        X_noisy = noisy_images.reshape(n_samples, n_features)

        psnr_list = []
        for r in r_values:
            # r不能超过样本数和特征数
            r_actual = min(r, n_samples, n_features)
            X_reconstructed = np.clip(pca_reconstruct(X_noisy, r_actual), 0, 1)

            # 计算所有样本的平均PSNR
            avg_psnr = np.mean([calculate_psnr(X_original[i], X_reconstructed[i])
                                for i in range(n_samples)])
            psnr_list.append(avg_psnr)

        results[sigma] = psnr_list
    return results


def plot_psnr_curves(r_values, results, save_path):  # 画PSNR曲线
    fig, ax = plt.subplots(figsize=(12, 8))

    colors = ['#E74C3C', '#E67E22', '#F1C40F', '#2ECC71', '#1ABC9C', '#3498DB', '#9B59B6']
    markers = ['o', 's', '^', 'D', 'v', 'p', 'h']

    for i, (sigma, psnr_list) in enumerate(results.items()):
        ax.plot(r_values[:len(psnr_list)], psnr_list,
                color=colors[i % len(colors)],
                marker=markers[i % len(markers)],
                markersize=6, linewidth=2,
                label=f'  = {sigma}', alpha=0.85)

    ax.set_xlabel('Number of Principal Components (r)', fontsize=14, fontweight='bold')
    ax.set_ylabel('PSNR (dB)', fontsize=14, fontweight='bold')
    ax.legend(loc='lower right', fontsize=11)
    ax.grid(True, linestyle='--', alpha=0.7)
    ax.set_xlim([0, max(r_values) + 5])

    plt.tight_layout()
    plt.savefig(save_path, dpi=300, bbox_inches='tight')
```

```
    plt.close()


def visualize_reconstruction(images, r, sigma, save_path):  # 可视化重建效果
    n_samples = images.shape[0]
    n_features = images.shape[1] * images.shape[2]
    image_shape = images.shape[1:]

    # 取第一张图做演示
    original_img = images[0]
    noisy_images = add_gaussian_noise(images, sigma)
    noisy_img = noisy_images[0]

    # PCA重建
    X_noisy = noisy_images.reshape(n_samples, n_features)
    X_reconstructed = np.clip(pca_reconstruct(X_noisy, min(r, n_samples, n_features)), 0, 1)
    reconstructed_img = X_reconstructed[0].reshape(image_shape)

    # 计算PSNR
    psnr_noisy = calculate_psnr(original_img.flatten(), noisy_img.flatten())
    psnr_reconstructed = calculate_psnr(original_img.flatten(), reconstructed_img.flatten())

    # 画三张图对比：原图、噪声图、重建图
    fig, axes = plt.subplots(1, 3, figsize=(15, 5))

    axes[0].imshow(original_img, cmap='gray', vmin=0, vmax=1)
    axes[0].set_title('Original', fontsize=14, fontweight='bold')
    axes[0].axis('off')

    axes[1].imshow(noisy_img, cmap='gray', vmin=0, vmax=1)
    axes[1].set_title(f'Noisy ( ={sigma})\nPSNR={psnr_noisy:.2f} dB', fontsize=14,
    fontweight='bold')
    axes[1].axis('off')

    axes[2].imshow(reconstructed_img, cmap='gray', vmin=0, vmax=1)
    axes[2].set_title(f'Reconstructed (r={r})\nPSNR={psnr_reconstructed:.2f} dB', fontsize
    =14, fontweight='bold')
    axes[2].axis('off')

    plt.tight_layout()
    plt.savefig(save_path, dpi=300, bbox_inches='tight')
    plt.close()


def main():
    # 加载数据
    images, _ = load_yalefaces('./yalefaces')
    images = normalize_images(images)
```

```
163
164     # 实验参数
165     r_values = list(range(5, 201, 5))  # 主成分数量: 5, 10, 15, ..., 200
166     sigma_values = [0.0001, 0.001, 0.01, 0.05, 0.1, 0.5, 1.0]  # 噪声强度
167
168     print("开始实验...")
169     results = run_experiment(images, r_values, sigma_values)
170
171     # 画图
172     plot_psnr_curves(r_values, results, 'psnr_curves.png')
173     print("PSNR曲线已保存")
174
175     visualize_reconstruction(images, r=20, sigma=0.05, save_path='
        reconstruction_r20_sigma005.png')
176     print("重建对比图已保存")
177
178
179 if __name__ == "__main__":
180     main()
```