



中山大學  
SUN YAT-SEN UNIVERSITY

## 模式识别

### 基于 PCA 的图像压缩

姓名 马岱

学号 22336180

学院 计算机学院

专业 计算机科学与技术

2025 年 5 月 16 日

# 目录

<b>1 实验目的</b>	<b>3</b>
<b>2 实验内容</b>	<b>3</b>
<b>3 实验过程</b>	<b>3</b>
3.1 算法原理 . . . . .	3
3.1.1 PCA (主成分分析) . . . . .	3
3.2 实验步骤 & 代码 . . . . .	6
3.2.1 实现 PCA 函数接口 . . . . .	6
3.2.2 灰度人脸数据压缩和重建 . . . . .	7
3.2.3 彩色 RGB 图的压缩和重建 . . . . .	8
<b>4 实验结果</b>	<b>9</b>
4.1 灰度人脸数据压缩和重建 . . . . .	9
4.1.1 结果展示 . . . . .	9
4.1.2 实验分析 . . . . .	11
4.2 彩色 RGB 图的压缩和重建 . . . . .	11
4.2.1 结果展示 . . . . .	11
4.2.2 实验分析 . . . . .	12
<b>5 实验总结</b>	<b>13</b>

# 1 实验目的

- 熟悉并掌握主成分分析的基本原理
- 学会应用主成分分析实现数据降维，并应用到图像压缩

# 2 实验内容

- 按照主成分分析的原理实现 PCA 函数接口
- 利用实现的 PCA 函数对图像数据进行压缩和重建
- 利用实现的 PCA 函数对高维数据进行低维可视化

# 3 实验过程

## 3.1 算法原理

### 3.1.1 PCA (主成分分析)

主成分分析 (Principal Component Analysis, PCA) 是一种用于高维数据降维的经典线性算法，其核心思想是通过寻找数据的正交方向，使得数据在这些方向上的投影能够最大限度地保留方差或最小化重构误差。以下从数学原理、推导及算法流程等方面进行详细说明。以下从数学原理、推导及算法流程等方面进行详细说明。

**数据预处理（去中心化）** 设原始数据矩阵为  $X \in \mathbb{R}^{n \times d}$ ，其中  $n$  为样本数， $d$  为特征维度。首先计算样本均值向量：

$$\bar{x} = \frac{1}{n} \sum_{i=1}^n x_i, \quad x_i \in \mathbb{R}^d.$$

然后对数据进行去中心化处理：

$$X_c = X - \mathbf{1}_n \bar{x}^T,$$

其中  $\mathbf{1}_n \in \mathbb{R}^n$  是元素全为 1 的列向量。

**计算协方差矩阵** 基于去中心化后的数据  $X_c$ ，协方差矩阵定义为：

$$C = \frac{1}{n-1} X_c^T X_c \in \mathbb{R}^{d \times d}.$$

矩阵  $C$  的对角线上元素为各维度的方差，非对角线元素为不同维度之间的协方差。

**特征分解与主成分选取** 对协方差矩阵  $C$  进行特征值分解：

$$C u_i = \lambda_i u_i, \quad i = 1, 2, \dots, d,$$

得到实数特征值  $\lambda_1 \geq \lambda_2 \geq \dots \geq \lambda_d \geq 0$  及对应单位正交特征向量  $\{u_i\}$ 。将特征向量按特征值从大到小排列，取前  $k$  个：

$$U_k = [u_1, u_2, \dots, u_k] \in \mathbb{R}^{d \times k}.$$

**数据降维投影** 将去中心化数据投影到主成分空间：

$$Z = X_c U_k \in \mathbb{R}^{n \times k},$$

其中每一行  $z_i = U_k^T(x_i - \bar{x})$  表示样本在前  $k$  个主成分上的坐标。

**数据重建（逆变换）** 基于降维表示  $Z$

$$\hat{X} = Z U_k^T + \mathbf{1}_n \bar{x}^T \in \mathbb{R}^{n \times d},$$

得到的  $\hat{X}$  即为从子空间重建回原始空间的近似数据。

## 数学推导

**最大化投影方差视角** 希望在一个单位向量  $\mathbf{u} \in \mathbb{R}^D$  上投影后，投影方差最大：

$$\max_{\mathbf{u}} \text{Var}(\mathbf{u}^\top \tilde{\mathbf{x}}) = \max_{\mathbf{u}} \mathbf{u}^\top S \mathbf{u}, \quad \text{s.t. } \mathbf{u}^\top \mathbf{u} = 1.$$

利用拉格朗日乘子法，构造：

$$L(\mathbf{u}, \lambda) = \mathbf{u}^\top S \mathbf{u} - \lambda(\mathbf{u}^\top \mathbf{u} - 1).$$

对  $\mathbf{u}$  求导并令其为零：

$$\frac{\partial L}{\partial \mathbf{u}} = 2S\mathbf{u} - 2\lambda\mathbf{u} = 0 \implies S\mathbf{u} = \lambda\mathbf{u}.$$

因此， $\mathbf{u}$  为  $S$  的特征向量，投影方差  $= \lambda$ 。取最大特征值对应的特征向量为第一主成分方向  $\mathbf{u}_1$ 。依次在剩余特征空间中取次大特征值对应的特征向量  $\mathbf{u}_2, \dots, \mathbf{u}_M$ 。

**最小化重构误差视角** 考虑将原始数据投影到  $M$  维子空间后重构，投影系数为  $\alpha_{i,j} = \mathbf{u}_j^\top \tilde{\mathbf{x}}_i$ ，重构为

$$\hat{\mathbf{x}}_i = \sum_{j=1}^M \alpha_{i,j} \mathbf{u}_j.$$

定义重构误差总和：

$$E = \sum_{i=1}^N \|\tilde{\mathbf{x}}_i - \hat{\mathbf{x}}_i\|^2 = \|X - UU^\top X\|_F^2,$$

其中  $U = [\mathbf{u}_1, \dots, \mathbf{u}_M] \in \mathbb{R}^{D \times M}$ ,  $U^\top U = I_M$ ,  $\|\cdot\|_F$  为 Frobenius 范数。可以证明该误差最小化问题等价于在约束条件下最大化上述投影方差，从而导出相同的特征分解解。

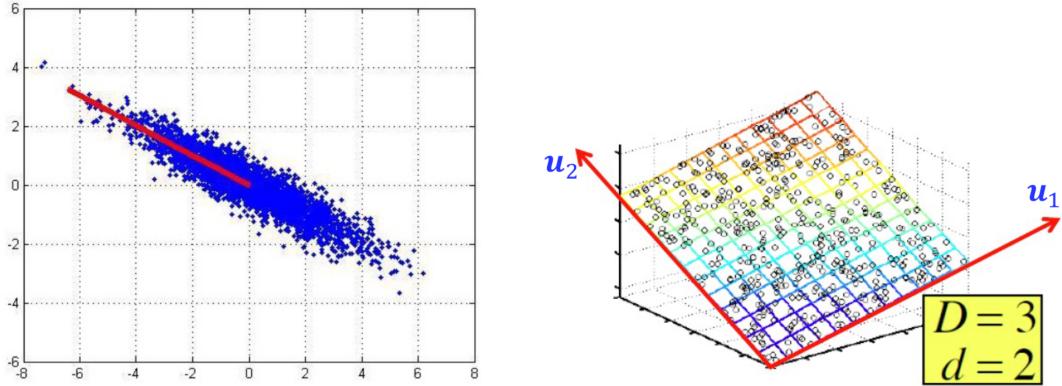


图 1: 左: 最大化投影方差, 右: 最小化重构误差

**SVD 视角** 对中心化数据矩阵  $A$  作奇异值分解:

$$A = U\Sigma V^T,$$

其中  $U \in \mathbb{R}^{D \times D}$  为左奇异向量,  $\Sigma \in \mathbb{R}^{D \times N}$  对角元素为奇异值,  $V \in \mathbb{R}^{N \times N}$  为右奇异向量。令  $\Sigma = [\text{diag}(\sigma_1, \dots, \sigma_r), \mathbf{0}]$ , 则协方差矩阵  $S = \frac{1}{N}AA^T = U(\frac{1}{N}\Sigma\Sigma^T)U^T$ 。故  $U$  的前  $M$  列即为 PCA 所需主成分方向。

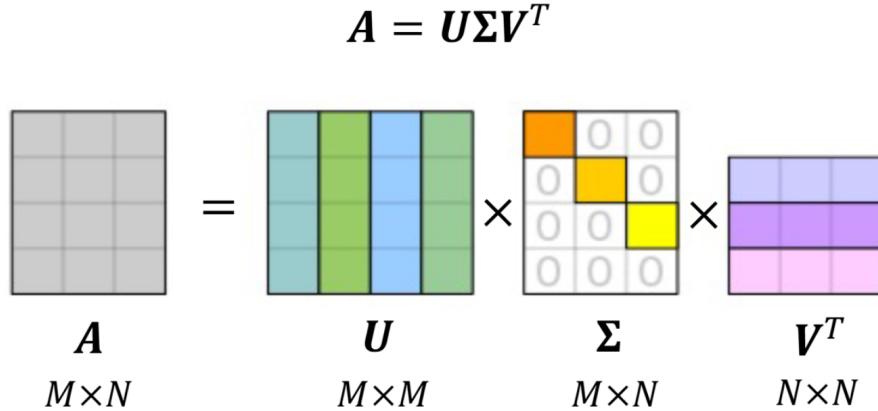


图 2: SVD 分解

## 算法流程

1. 给定原始数据集  $\{\mathbf{x}_i\}_{i=1}^N$ , 计算均值并中心化。
2. 构造中心化数据矩阵  $X$ , 计算协方差矩阵  $S = \frac{1}{N}XX^T$ 。
3. 求解  $S$  的特征值分解  $S = W\Lambda W^T$ , 令  $W = [\mathbf{w}_1, \dots, \mathbf{w}_D]$ ,  $\Lambda = \text{diag}(\lambda_1, \dots, \lambda_D)$ , 且  $\lambda_1 \geq \dots \geq \lambda_D$ 。

4. 取前  $M$  个最大特征值对应的特征向量  $\{\mathbf{w}_1, \dots, \mathbf{w}_M\}$ , 构成投影矩阵  $U_M \in \mathbb{R}^{D \times M}$ 。
5. 对新样本  $\mathbf{x}$ , 先中心化再投影:

$$\mathbf{z} = U_M^\top (\mathbf{x} - \bar{\mathbf{x}}),$$

得到  $M$  维低维表示。

## 注意事项与扩展

- 特征值快速衰减时, 可用较少分量获得良好表示。
- 若  $D \gg N$ , 直接对  $X^\top X$  进行特征分解更高效。
- 可以用于噪声去除、数据压缩、可视化等任务。

## 3.2 实验步骤 & 代码

根据上一小节的算法原理, 我们在这一部分直接给出相关的代码和解释。

### 3.2.1 实现 PCA 函数接口

PCA 函数的主要流程是: 先对计算数据的协方差矩阵, 然后在对协方差矩阵进行 SVD 分解, 得到对应的特征值和特征向量。

我实现了一个 PCA 的类, 其接收一个整型参数 `n_components`, 代表所保留的主成分个数。类内还定义了三个成员变量, 分别是: `mean_`, 用于记录数据的均值向量; `components_`, 用于记录降维后的特征向量矩阵(即主成分); 以及 `explained_variance_`, 用于存储对应主成分的特征值大小。

其中 `fit` 方法首先计算每列特征的均值, 并用该均值对数据进行去中心化, 结果保存在 `X_centered` 中。随后计算去中心化后数据的协方差矩阵, 利用 `np.dot(X_centered.T, X_centered)` 得到协方差矩阵的无偏估计。接着对协方差矩阵进行奇异值分解 (SVD), 提取左奇异矩阵 `U` 和奇异值 `S`, 其中左奇异矩阵的列向量即为特征空间的正交基。最后, 保留前 `n_components` 个主成分, 将其存入 `components_` 中, 对应的奇异值存入 `explained_variance_` 中。

`transform` 方法则用于将原始数据 `X` 投影到之前通过 `fit` 方法学到的低维特征空间。首先将输入数据进行去中心化, 然后将去中心化后的数据与主成分矩阵相乘, 得到在新特征空间下的低维表示矩阵。

`inverse_transform` 方法用于将低维空间中的数据恢复到原始空间。通过将低维数据与主成分矩阵的转置相乘, 再加上均值向量, 即可得到重建后的数据矩阵。虽然重建后的数据不完全等于原始数据, 但能够尽量保留数据的主要特征信息。

```

1  class PCA:
2      """
3          自实现 PCA 类，用于降维和重建
4      """
5
6      def __init__(self, n_components: int):
7          self.n_components = n_components
8          self.mean_ = None
9          self.components_ = None
10         self.explained_variance_ = None
11
12     def fit(self, X: np.ndarray):
13         # 去中心化
14         self.mean_ = np.mean(X, axis=0)
15         X_centered = X - self.mean_
16         # 协方差矩阵
17         cov = np.dot(X_centered.T, X_centered) / (X_centered.shape[0] - 1)
18         # SVD 分解
19         U, S, _ = np.linalg.svd(cov)
20         # 选取前 n_components
21         self.components_ = U[:, : self.n_components]
22         self.explained_variance_ = S[: self.n_components]
23         return self
24
25     def transform(self, X: np.ndarray) -> np.ndarray:
26         X_centered = X - self.mean_
27         return np.dot(X_centered, self.components_)
28
29     def inverse_transform(self, Z: np.ndarray) -> np.ndarray:
30         X_rec = np.dot(Z, self.components_.T) + self.mean_
31         return X_rec
32
33     def fit_transform(self, X: np.ndarray) -> np.ndarray:
34         self.fit(X)
35         return self.transform(X)

```

### 3.2.2 灰度人脸数据压缩和重建

首先构建数据与结果路径，并调用 `load_faces` 获取形状为  $(n\_samples, 1024)$  的人脸矩阵，再由特征维度推断单张图像的边长  $\sqrt{1024} = 32$ 。接着脚本将前 49 张原始人脸以  $7 \times 7$  网格保存为 `eigen_faces.jpg`，方便直观查看原始样本。随后针对不同主成

分数  $k \in \{10, 50, 100, 150\}$ , 脚本实例化 PCA 对象并使用 `fit_transform` 得到低维表示, 再通过 `inverse_transform` 将其重建回高维, 最后将前 49 张重建图像同样以  $7 \times 7$  网格保存为 `recovered_faces_top_{k}.jpg`。

```

1 def main():
2     data_path = os.path.join("data", "faces.mat")
3     out_dir = os.path.join("results", "PCA")
4     os.makedirs(out_dir, exist_ok=True)
5
6     # 1. 加载人脸数据
7     faces = load_faces(data_path) # (n_samples, 1024)
8     n_samples, n_features = faces.shape
9     side = int(np.sqrt(n_features))
10    assert side * side == n_features, f"不能将 {n_features} 重塑为正方形"
11
12    # 2. 保存前 49 张原始人脸 (7*7 网格)
13    save_grid(
14        faces[:49], (7, 7), (side, side), os.path.join(out_dir,
15                           "eigen_faces.jpg"))
16
17    # 3. 对不同 k 值进行压缩和重建, 保存前 49 张重建结果 (7x7 网格)
18    for k in [10, 50, 100, 150]:
19        pca = PCA(n_components=k)
20        Z = pca.fit_transform(faces) # (n_samples, k)
21        faces_rec = pca.inverse_transform(Z) # (n_samples, 1024)
22        save_grid(
23            faces_rec[:49],
24            (7, 7),
25            (side, side),
26            os.path.join(out_dir, f"recovered_faces_top_{k}.jpg"),
27        )

```

### 3.2.3 彩色 RGB 图的压缩和重建

利用实现的 PCA 函数, 对 example.png 彩色 RGB 图进行压缩和重建。数据位于 `data/woman.jpg`, 对该图片分布降维到不同维度 (10, 50, 100, 150) 进行压缩, 然后再重建, 对比不同的压缩和重建效果。将结果保存为 `results/PCA/recovered_woman_top_xxx.jpg`。实验报告中要有压缩前, 和不同压缩程度的结果对比。

首先从输入的三维图像数组中获取高度、宽度和通道数, 然后创建一个相同形状但

类型为 float64 的空数组用于存放重建结果。随后，函数依次对 R、G、B 三个通道进行处理：它将当前通道的数据转换到浮点型，并视作一个大小为  $H \times W$  的二维矩阵，然后实例化  $\text{PCA}(\text{n\_components}=k)$ ，调用 `fit_transform` 方法得到降维后的表示矩阵  $Z \in \mathbb{R}^{H \times k}$ ，接着通过 `inverse_transform` 将这些低维表示映射回原始空间得到重建后的通道矩阵，最后将该矩阵存入对应的通道位置。当所有通道处理完毕后，函数对重建结果进行裁剪，确保像素值在 [0, 255] 范围内，并将数组转换为 uint8 类型返回。

```

1 def compress_and_reconstruct_image(img_array: np.ndarray, k: int) ->
2     np.ndarray:
3         """
4             对一张 RGB 图像的每个通道分别执行 PCA 压缩和重建，并将三个通道合并。
5             img_array: 原始图像数组, shape=(H, W, 3), dtype=np.uint8
6             k:      降到的主成分数量
7             返回重建后的图像数组, shape 同原图, dtype=np.uint8
8         """
9
10    H, W, C = img_array.shape
11    reconstructed = np.zeros_like(img_array, dtype=np.float64)
12    # 对 R/G/B 三个通道分别处理
13    for ch in range(C):
14        channel = img_array[:, :, ch].astype(np.float64)
15        # 将二维图像展平为一维向量集合: shape=(H, W) ->(H, W) 本身即可视为
16        # n_samples=H, n_features=W
17        # 若想按整图做 PCA, 可 reshape 为 (H*W, 1) ——但这里我们按行做降维重建
18        # 更常见做法: 把图像当作 n_samples=H, n_features=W; 重建后再拼回
19        pca = PCA(n_components=k)
20        Z = pca.fit_transform(channel) # shape=(H, k)
21        rec_channel = pca.inverse_transform(Z) # shape=(H, W)
22        reconstructed[:, :, ch] = rec_channel
23        # 裁剪并转换为 uint8
24        reconstructed = np.clip(reconstructed, 0, 255).astype(np.uint8)
25    return reconstructed

```

## 4 实验结果

### 4.1 灰度人脸数据压缩和重建

#### 4.1.1 结果展示

原始图像和压缩重构的图像如下所示：

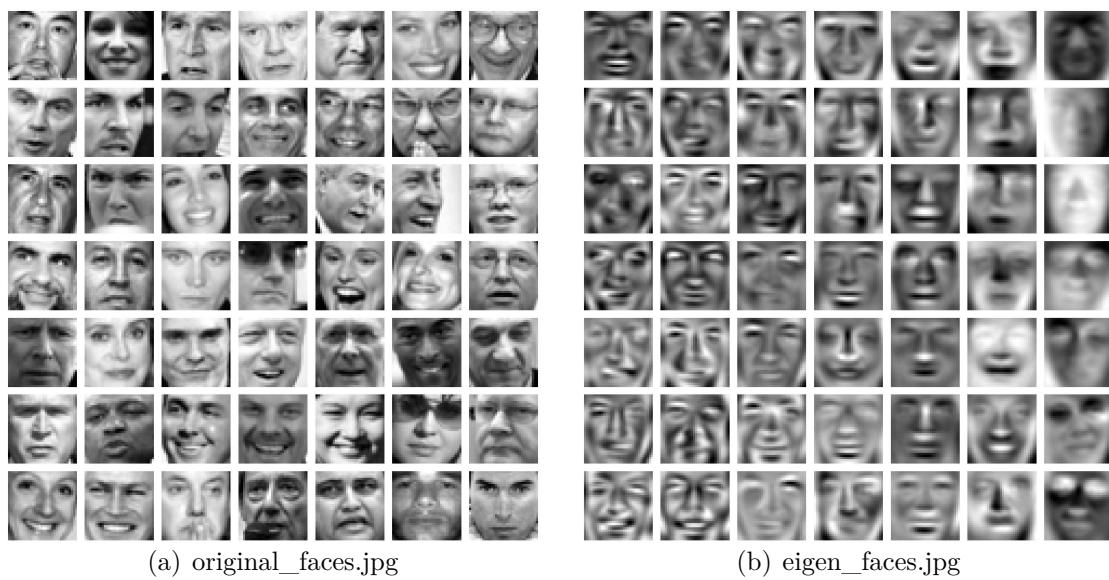


图 3: 不同压缩维度下的重建图像 (依次为 10, 50, 100, 150)

### 4.1.2 实验分析

在对 EigenFace 数据集中的灰度人脸进行 PCA 压缩与重建实验中，我们首先观察原始前 49 张人脸图像，它们包含了完整的面部细节与纹理信息。当仅保留前 10 个主成分进行重建时，重建图像轮廓仍能大致辨认出眼睛、鼻子和脸型的大致位置，但表情细节、光影变化以及面部特征的清晰度已经大幅丢失，图像呈现明显的模糊与块状色块。随着主成分数增至 50，重建结果开始恢复更多的面部细节，眼眶与鼻唇部位的阴影渐趋自然，脸部轮廓也较为平滑，说明此时所用主成分已捕捉到大部分典型的人脸变化模式。进一步将主成分数量提升到 100 时，重建图像中的细微特征（如嘴角的轻微上扬、额头的光影过渡）都得到了较好的还原，整体视觉效果与原图十分接近，仅在极细微的噪声和高频细节处略有差异。当主成分数量达到 150 时，几乎无法用肉眼区分重建图与原始图的差别，表明此时绝大部分信息已被前 150 个特征向量所涵盖。整体来看，本实验验证了 PCA 在人脸压缩中的有效性：较少的主成分可大幅降低数据维度和存储成本，但会造成明显模糊；而适当提高主成分数量，则能在保证高压缩比的同时，较好地保留面部特征信息。在实际应用中，需要根据存储、传输效率与重建质量的需求，在主成分数量和压缩率之间权衡选择，以获得最佳的压缩效果。

## 4.2 彩色 RGB 图的压缩和重建

### 4.2.1 结果展示



图 4: 原始图像 woman\_original.jpg



图 5: 不同压缩维度下的重建图像 (依次为 10, 50, 100, 150)

#### 4.2.2 实验分析

通过对原始彩色图像进行 PCA 压缩和重建实验，可以明显观察到随着主成分数量的增加，重建图像的质量逐渐提升。当压缩维度为 10 时，图像细节严重缺失，仅能勉强辨认出轮廓与大致颜色分布，图像出现明显的模糊与失真现象。将维度提升至 50 后，图像的轮廓与主要区域特征得以保留，尽管细节部分仍存在一定程度的模糊，但整体的结构和色彩分布已经较为自然。当维度增加到 100 时，重建图像与原图基本接近，图像的纹理、轮廓以及色彩还原效果良好，仅在个别细节处略有差异。最终，当维度为 150 时，重建图像几乎与原始图像无异，人眼难以察觉差别，表明此时大部分原始信息已经通过主成分成功保留。实验结果说明，PCA 在图像压缩中的效果与主成分数量密切相关，较少的主成分虽然能显著减少数据量，但也会导致图像细节严重缺失，而适当增加主成分数量能够在保证压缩率的同时有效保留图像的主要信息和视觉质量。因此，在实际应用中，应根据对压缩率与图像质量的需求，合理选择主成分数量，以达到最佳平衡。

## 5 实验总结

本次实验围绕自实现的 PCA 算法对两类图像数据——EigenFace 人脸集与彩色 woman.jpg——进行了系统的降维压缩与重建对比，验证了 PCA 在图像特征提取和压缩还原中的卓越作用。首先，对 EigenFace 数据集进行灰度人脸处理，我们加载并展示了前 100 张原始样本，又提取了前 49 个主成分形成“特征脸”网格，并分别以  $k=10$ 、 $50$ 、 $100$ 、 $150$  进行降维重建。结果表明，仅保留 10 个主成分时，人脸重建仅呈现模糊轮廓；50 个主成分可还原五官结构与阴影层次；100 个主成分时，细节与纹理高度近似原图；150 个主成分几乎完美复现。其次，对彩色图像 woman.jpg，我们将每个 RGB 通道单独做 PCA 降维重建， $k$  同样取  $10$ 、 $50$ 、 $100$ 、 $150$ 。实验结果显示，10 维重建色块明显、细节全失；50 维时主要色彩层次与边缘轮廓可辨；100 维下色彩过渡与纹理恢复良好；150 维几近无差异。两部分实验的对比一致地说明：PCA 能以极少的主成分保留图像核心信息，随着主成分数量增加，重建质量显著提升，而最优的压缩比与视觉效果平衡则取决于应用场景对存储效率与图像精度的需求。