统计学习与模式识别实验一：PCA 与 SVD 的图像压缩

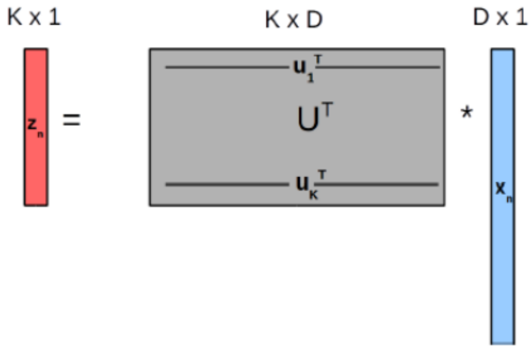
蔡与望　2020010801024

一　实验原理

1.1　PCA 概述

填满空间的样本随着空间的维度而指数增长，造成“维度诅咒”。因此，我们需要一种方法降维，减少过拟合概率，增强泛化能力。主成分分析（PCA）就是一种常用的降维方法。

PCA 可以从样本数据中学习一种投影方向，使得投影方向上样本的方差最大，或者说重构的误差最小。这可以具象为一个投影矩阵U：对于一个$D \times N$维的数据矩阵，PCA 可以使用一个$D \times K (D > K)$维的投影矩阵U，将原来的数据投影到K维上。



具体的算法是：

1. 数据中心化，每个样本都减去样本均值。（$\mu = \frac{1}{N}\sum_{i=1}^{N} x_i$）
2. 计算中心化样本的协方差矩阵。（$S = \frac{1}{N} X X^T$）
3. 对协方差矩阵进行特征分解。
4. 将特征值升序排列，取前若干个对应的特征向量$\{u_k\}$。
5. 对样本进行投影。（$Z = U^T X$）

1.2　PCA 图像压缩

我们读取图像后，可以在 RGB 三个通道的像素点上，分别提取主成分。

如果我们使用所有主成分重构，那么得出的图像就是原来的图像。但为了压缩的目的，我们可以使用前若干个主成分，例如前 50 个，近似重构图像。

一些图像的细节会丢失；但由于主成分标志着最小重构误差的方向，所以图像大体上不会有差异。

至于具体使用多少个主成分，我们可以绘制累计贡献率的碎石图，取若干个阈值（例如 30%、50%、70%、90%、99%），分别观察所需主成分的个数。可以预见的是，累计贡献率越高，主成分个数越多，计算越复杂，得到的压缩效果也越好。

最后，我们将这三个重构后的通道合并，将得到的图像渲染出来，就能够预览压缩的效果。

1.3　SVD 概述

SVD 的核心观点是：任何一个矩阵造成的变换，都可以被分解为一个旋转、放缩、再

旋转的过程。

在数学表示上，就有样本矩阵X的 SVD 分解是$X = U\Sigma V^T$，其中U是所有左奇异列向量的并列（旋转），Σ是各奇异值排列成的对角矩阵（放缩），V是所有右奇异列向量的并列（再旋转）。

从另一个角度上来说，SVD 将一个矩阵分解成了若干个秩为 1 的矩阵的和。

1.4 SVD 图像压缩

如上所说，SVD 将矩阵分解成了若干个秩为 1 的矩阵的和。那么，我们就可以使用这些矩阵中的前若干个，来近似原矩阵。

我们同样需要将读取的图像分 RGB 三个通道，分别进行 SVD 分解。如果我们使用所有的奇异值，那么最后得到的就是原图像。为了压缩，我们可以各选择U、Σ、V的前若干个，例如前 50 个，近似重构图像。

对于 SVD，我们没有类似于 PCA 中“累计贡献率”那么方便的衡量指标，但我们可以采用奇异值累计和占奇异值总和的比例，来近似地衡量多少个奇异值能取得较好的压缩效果。例如，前 50 个奇异值占了总和的 90%，那么它们的重构就应该是一个较好的图像压缩。

最后，我们将重构的通道合并、渲染，就能够预览压缩的效果。

二 代码实现

2.1 PCA

手动实现一个简易的 PCA，与 scikit-learn 中的 PCA 保持相同的 API。

```
class PCA:
    def __init__(self, n_components=None):
        self.n_components = n_components
        self.mean = None
        self.components = None
        self.explained_variance_ratio = None

    def fit(self, data):
        self.mean = np.mean(data, axis=0)
        centered_data = data - self.mean

        cov = np.cov(centered_data.T)
        eigenvalues, eigenvectors = np.linalg.eig(cov)

        sorted_indices = np.argsort(eigenvalues)[::-1]
        eigenvalues = eigenvalues[sorted_indices]
        eigenvectors = eigenvectors[:, sorted_indices]

        self.components = eigenvectors[:, :self.n_components]
        self.explained_variance_ratio = eigenvalues / np.sum(eigenvalues)

    def transform(self, data):
        centered_data = data - self.mean
        return np.dot(centered_data, self.components)

    def fit_transform(self, data):
        self.fit(data)
        return self.transform(data)

    def inverse_transform(self, pca_data):
        return np.dot(pca_data, self.components.T) + self.mean
```

读取图像，分别提取出 RGB 三个通道的数据，并分析它们的主成分。

取 0.5、0.9、0.99、0.999 四个累计贡献率阈值，观察需要多少个主成分的共同贡献，才能达到这个精度。

```
image = plt.imread("butterfly.bmp")

red_component = image[:, :, 0]
green_component = image[:, :, 1]
blue_component = image[:, :, 2]
```

```python
precision_list = (0.3, 0.5, 0.7, 0.9, 0.99)

fig, axes = plt.subplots(1, 3, figsize=(15, 5))
fig.suptitle("主成分数-累计贡献率")

def plot_accumulative_contribution(color):
    config = configs[color]
    title = config["title"]
    color = config["color"]
    data = config["data"]
    ax = axes[config["axis"]]

    pca = PCA()
    pca.fit(data)
    accumulative_contribution = np.cumsum(pca.explained_variance_ratio)
    ax.plot(accumulative_contribution, color=color)
    ax.set_title(title)

    for precision in precision_list:
        component_num = len(
            accumulative_contribution[accumulative_contribution < precision]
        )
        ax.plot(component_num, precision, color=color, marker="o")
        ax.text(component_num + 5, precision - 0.04, f"({component_num},{precision})")

plot_accumulative_contribution("red")
plot_accumulative_contribution("green")
plot_accumulative_contribution("blue")
plt.show()
```

挑选落在精度阈值上的主成分数，分别观察压缩效果。

```python
fig, axes = plt.subplots(2, 3, figsize=(15, 5))
fig.suptitle("压缩效果对比图")

n_components_list = (2, 5, 11, 27, 94)

for index, n_components in enumerate(n_components_list):
    pca = PCA(n_components)
    compressed = np.dstack((
        pca.inverse_transform(pca.fit_transform(red_component)),
        pca.inverse_transform(pca.fit_transform(green_component)),
        pca.inverse_transform(pca.fit_transform(blue_component)),
    )).astype(np.uint8)
    ax = axes[index // 3, index % 3]
    ax.imshow(compressed)
    ax.set_title(f"主成分数={n_components}, 精度={precision_list[index]}")
    ax.set_xticks([])
    ax.set_yticks([])

axes[1][2].imshow(image)
axes[1][2].set_title("原图")
axes[1][2].set_xticks([])
axes[1][2].set_yticks([])
plt.show()
```

2.2 SVD

读取图像，分别提取出 RGB 三个通道的数据，并对它们进行 SVD 分解。

取 0.5、0.9、0.99、0.999 四个累计贡献率阈值，观察需要多少个奇异值的共同贡献，才能达到这个精度。

```python
def plot_accumulative_contribution(color):
    config = configs[color]
    title = config["title"]
    color = config["color"]
    data = config["data"]
    ax = axes[config["axis"]]

    _, s, _ = np.linalg.svd(data)
    accumulative_contribution = np.cumsum(s) / np.sum(s)

    ax.plot(accumulative_contribution, color=color)
    ax.set_title(title)

    for precision in precision_list:
        component_num = len(
            accumulative_contribution[accumulative_contribution < precision]
        )
        ax.plot(component_num, precision, color=color, marker="o")
        ax.text(component_num + 5, precision - 0.04, f"({component_num},{precision})")

plot_accumulative_contribution("red")
plot_accumulative_contribution("green")
plot_accumulative_contribution("blue")
plt.show()
```

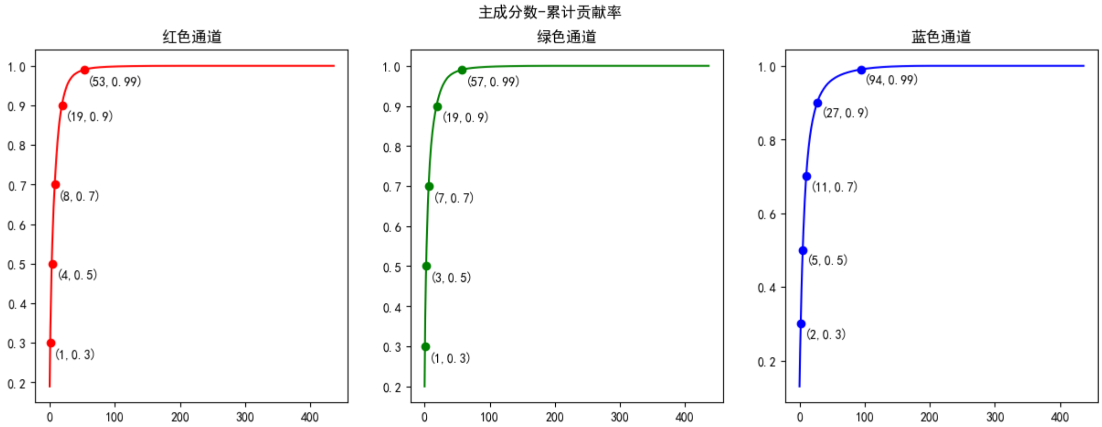
挑选落在阈值上的秩数，观察压缩效果。

```python
def compress_component(component, k):
    u, s, vh = np.linalg.svd(component)
    return (u[:, :k] @ np.diag(s[:k])) @ vh[:k]

fig, axes = plt.subplots(2, 3, figsize=(15, 5))
fig.suptitle("压缩效果对比图")

rank_list = (4, 13, 34, 95, 176)

for index, rank in enumerate(rank_list):
    compressed = np.dstack((
        compress_component(red_component, rank),
        compress_component(green_component, rank),
        compress_component(blue_component, rank),
    )).astype(np.uint8)
    ax = axes[index // 3, index % 3]
    ax.imshow(compressed)
    ax.set_title(f"秩={rank}，精度={precision_list[index]}")
    ax.set_xticks([])
    ax.set_yticks([])

axes[1][2].imshow(image)
axes[1][2].set_title("原图")
axes[1][2].set_xticks([])
axes[1][2].set_yticks([])
plt.show()
```

三 结果分析

3.1 PCA

从下图可以看到，主成分的累计贡献率一开始随着主成分数的增加而快速上升，红色和绿色通道在 50 多个主成分时达到 99%，蓝色则需要 94 个。



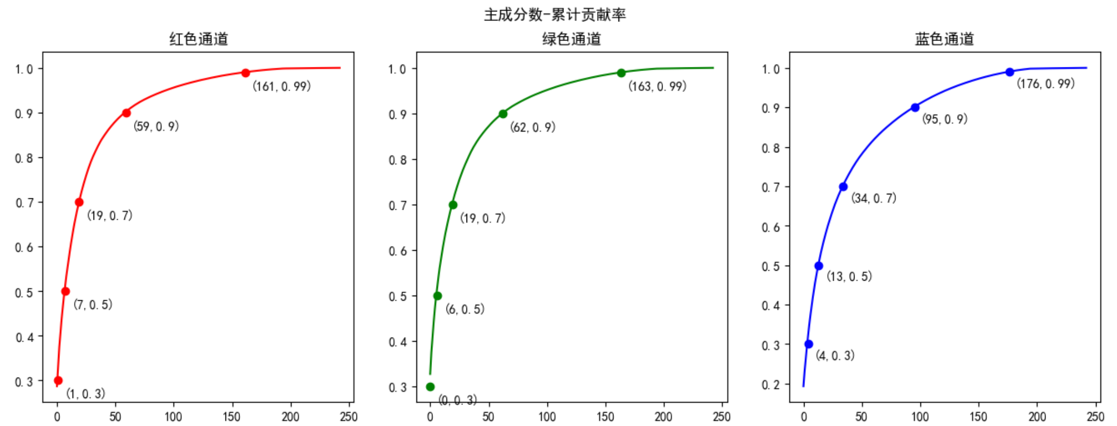
为了压缩的质量，我们以蓝色为准，分别选择 2、5、11、27、94 这五个主成分数，分别观察它们的压缩效果。

可以看到，随着主成分数的增加，压缩的效果也逐步提升。当主成分数较少，重构效果差，压缩后的图像十分模糊。当主成分数较多，图像的清晰度显著增加。这说明 PCA 的

图像压缩是成功的。

3.2 SVD

从下图可以看到，奇异值的累计贡献率一开始随着秩数的增加而快速上升，在 170 个左右时达到 99%。



为了压缩的质量，我们以蓝色为准，分别选择 4、13、34、95、176 这五个秩数，分别观察它们的压缩效果。

可以看到，随着秩数（奇异值数）的增加，压缩的效果也逐步提升。当秩数较少，重构效果差，压缩后的图像十分模糊。当秩数较多，图像的清晰度显著增加。这说明 SVD 的图像压缩也是成功的。

四　总结体会

通过本次实验，我对 PCA 和 SVD 的原理有了深入的了解。从本质上说，PCA 是对最大方差方向的计算，SVD 是对任意矩阵施加效果的旋转放缩分解；从图像压缩的角度来说，PCA 是对最小重构误差方向的计算，SVD 能够将矩阵分解成若干个 1-秩矩阵。

我对 PCA 和 SVD 在实际问题（图像处理）上的应用有了深刻的体会，同时也学会了使用 Python 处理图像的方法。