基于 PCA 的图像压缩

实验目的

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

实验要求

1. 实验报告要求有适当步骤说明和结果分析，与源代码打包提交
2. 编程语言可选择 python、matlab 等

实验内容

1. 按照主成分分析的原理实现 PCA 函数接口（30 分）
2. 利用 PCA 函数对图像数据进行压缩和重建，展示结果对比（50 分）
3. 将 PCA 降维应用到手写数字识别任务上（20 分）

实验过程

一．实现 PCA 函数接口

实现一个你自己的 PCA 函数。PCA 函数的主要流程是：先将原始数据标准化，再计算数据的协方差矩阵，然后对协方差矩阵进行 SVD 分解，得到对应的特征值和特征向量。

PCA 求解步骤

设有 m 条 n 维数据。

1. 将原始数据按列组成 n 行 m 列矩阵 X;
2. 将 X 的每一行进行零均值化，即减去这一行的均值;
3. 求出协方差矩阵 $C = \frac{1}{m} X X^{\mathsf{T}}$;
4. 求出协方差矩阵的特征值及对应的特征向量;
5. 将特征向量按对应特征值大小从上到下按行排列成矩阵，取前 k 行组成矩阵 P;
6. $Y = PX$ 即为降维到 k 维后的数据。

函数如下：

```
def my_pca(X, n_components):
    # 数据标准化
    X_std = X - np.mean(X, axis=0)

    # 计算协方差矩阵
    cov_matrix = np.cov(X_std, rowvar=False)
```

```
    # SVD 分解
    _, _, Vt = np.linalg.svd(cov_matrix)

    # 选择主成分
    components = Vt[:n_components]

    # 数据投影
    projected_data = np.dot(X_std, components.T)

    return projected_data, components
```

使用了numpy实现PCA函数

二．PCA 的基本应用

(a) 利用实现的 PCA 函数（调用库函数将无法得到步骤一的分数），对 EigenFace数据集中的灰度人脸数据进行压缩和重建。数据位于data/faces.mat，包含 5000 张 32×32 的灰度图像。采用 PCA 对这些人脸数据降维到不同维度(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/recovered_faces_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，定量对比要求计算峰值信噪比 PSNR。
首先导入数据：

```
data = scipy.io.loadmat('data/faces.mat')
faces_data = data['X']
```

然后通过PLT库函数保存原始图像，后面的保存重建后的图像也是一样的操作

```
for i in range(5):
        plt.imshow(faces_data[i].reshape(32, 32).T, cmap='gray')
        plt.axis('off')
        # plt.title(f'Original Face {i+1}')
        plt.savefig(f'results/original_face_{i+1}.jpg')
        plt.close()
# 创建一个空白的大图用于合成原始图像
big_original_image = Image.new('RGB', (320, 320))

# 设置每个小图像的位置和尺寸，并添加到大图上
for i in range(100):
    col = i % 10
    row = i // 10
    extent = (col * 32, row * 32, (col + 1) * 32, (row + 1) * 32)
    img_data = faces_data[i].reshape((32, 32)).T  # 转置矩阵以正确显示图像
    img_data = (img_data - np.min(img_data)) / (np.max(img_data) -
np.min(img_data))  # 将数据归一化到0-1范围
    img_data = (img_data * 255).astype(np.uint8)  # 转换为8位无符号整数
    original_image = Image.fromarray(img_data)
    big_original_image.paste(original_image, extent)
```

通过以下代码进行PCA压缩与重建：

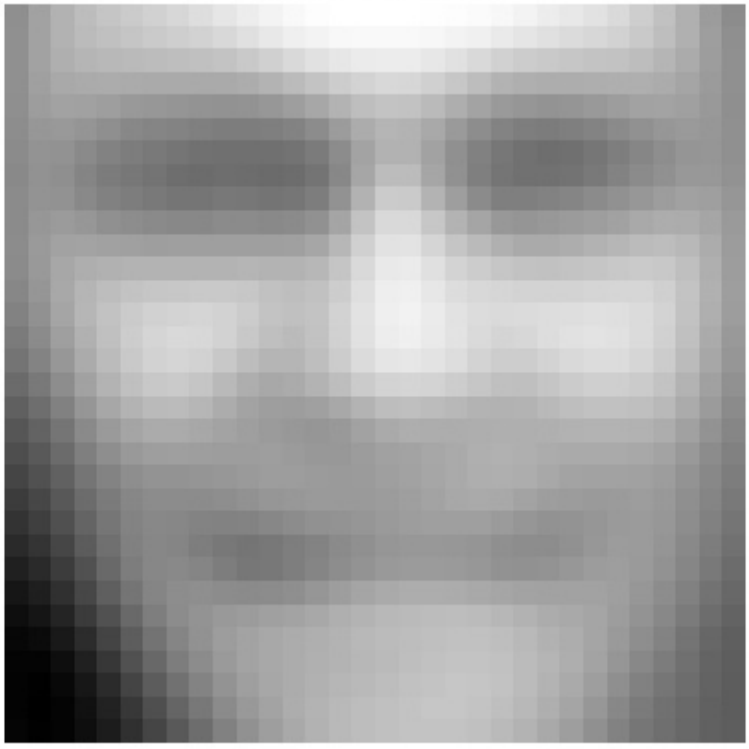
```
projected_data, components = my_pca(faces_data, dim)

# 重建数据
reconstructed_data = np.dot(projected_data, components)
reconstructed_data = reconstructed_data + np.mean(faces_data, axis=0)
```

结果展示：
由于图片太多，所以这里只选择前5张图像以及100张的合成大图进行展示
压缩前：

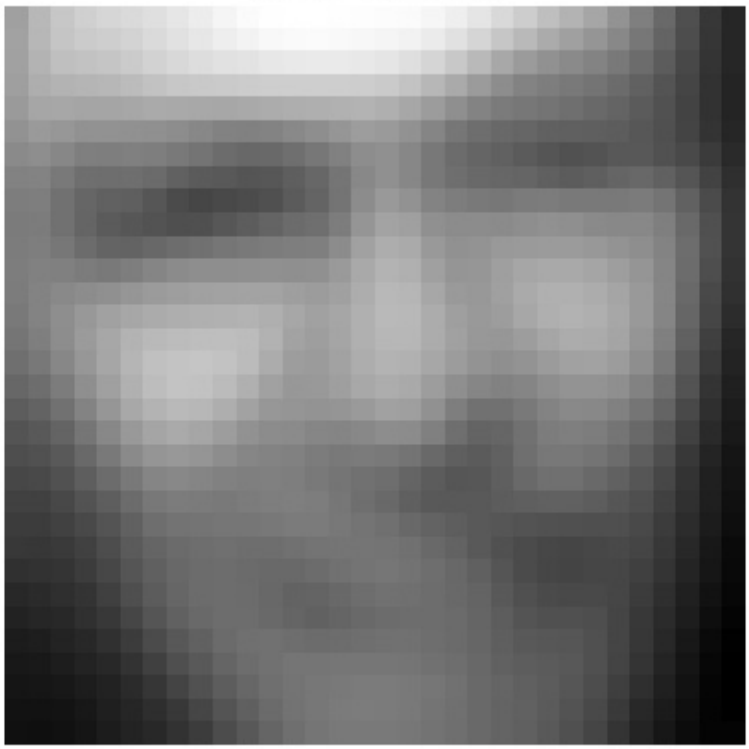
压缩

维度为10：

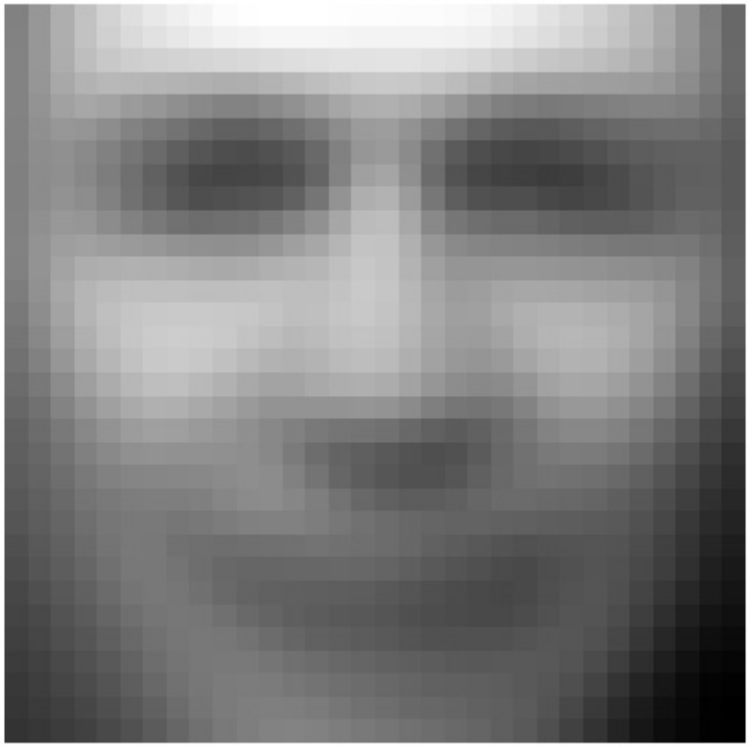
Compressed and Reconstructed Face 1 (Top 10)
PSNR: 20.22 dB



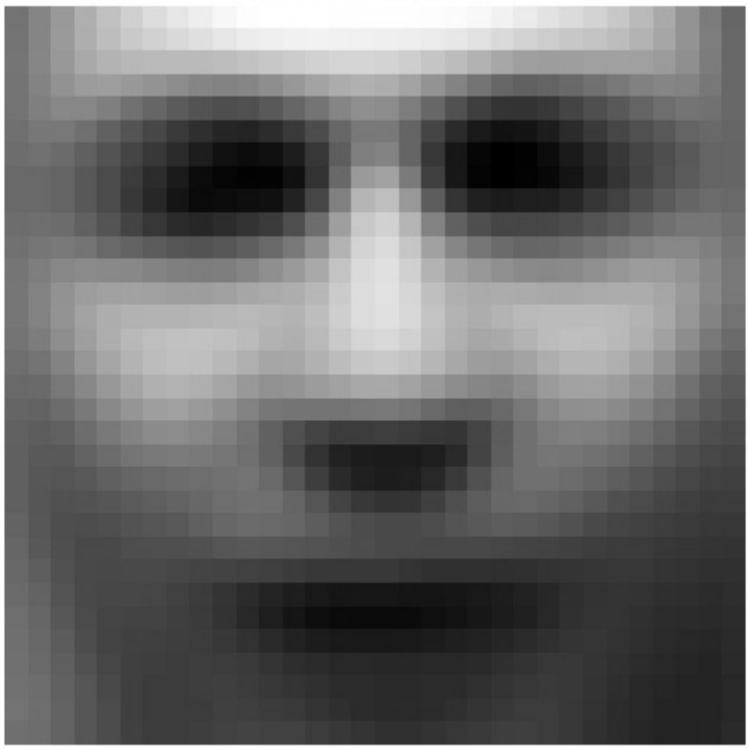
Compressed and Reconstructed Face 2 (Top 10)
PSNR: 23.87 dB

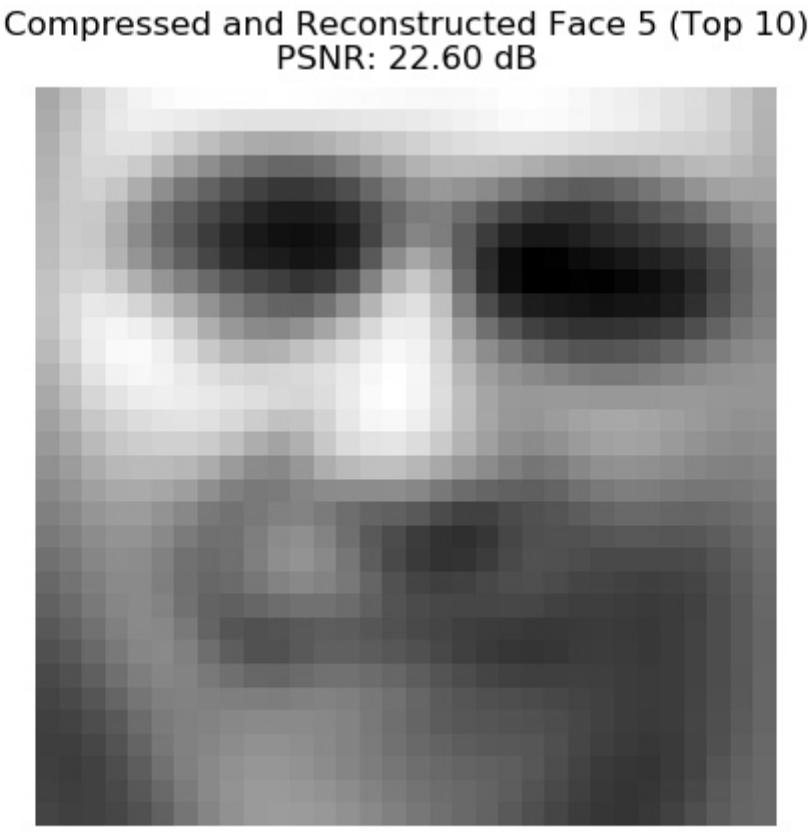


Compressed and Reconstructed Face 3 (Top 10)
PSNR: 19.97 dB



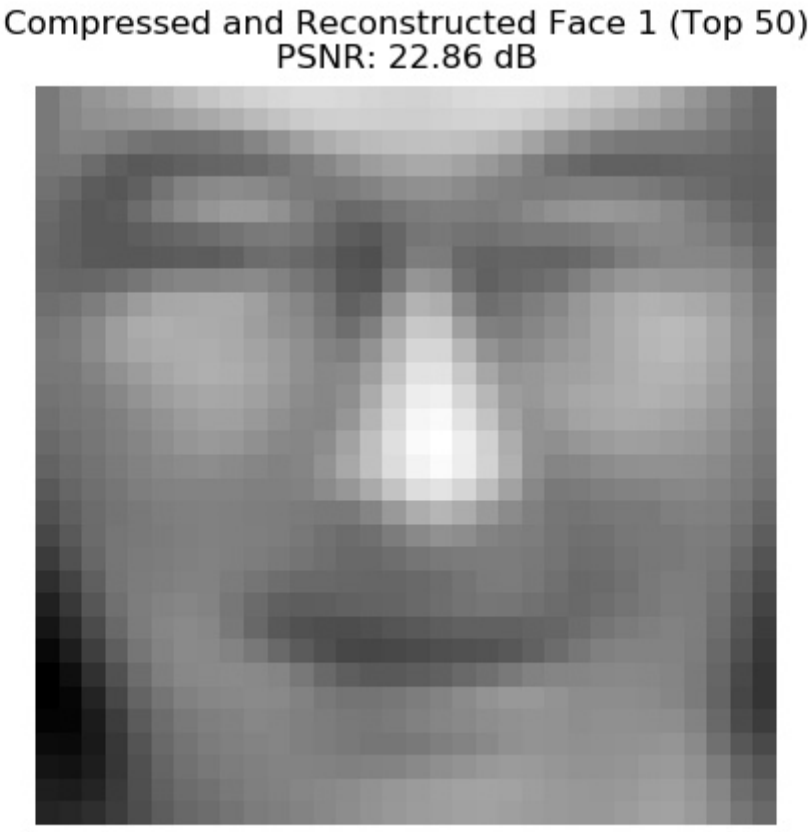
Compressed and Reconstructed Face 4 (Top 10)
PSNR: 23.70 dB




Compressed and Reconstructed Face 5 (Top 10)
PSNR: 22.60 dB

压缩

维度为50：


Compressed and Reconstructed Face 1 (Top 50)
PSNR: 22.86 dB

Compressed and Reconstructed Face 2 (Top 50)
PSNR: 27.56 dB

Compressed and Reconstructed Face 3 (Top 50)
PSNR: 25.25 dB

Compressed and Reconstructed Face 4 (Top 50)
PSNR: 28.64 dB

Compressed and Reconstructed Face 5 (Top 50)
PSNR: 26.68 dB

压缩

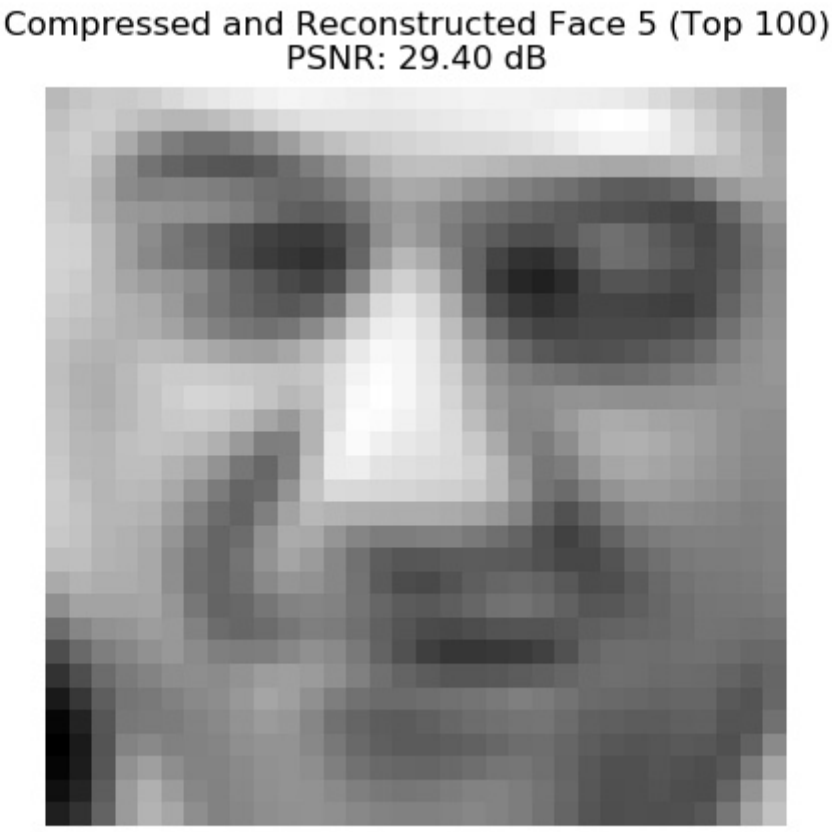
维度为100：

Compressed and Reconstructed Face 1 (Top 100)
PSNR: 25.75 dB

Compressed and Reconstructed Face 2 (Top 100)
PSNR: 31.59 dB

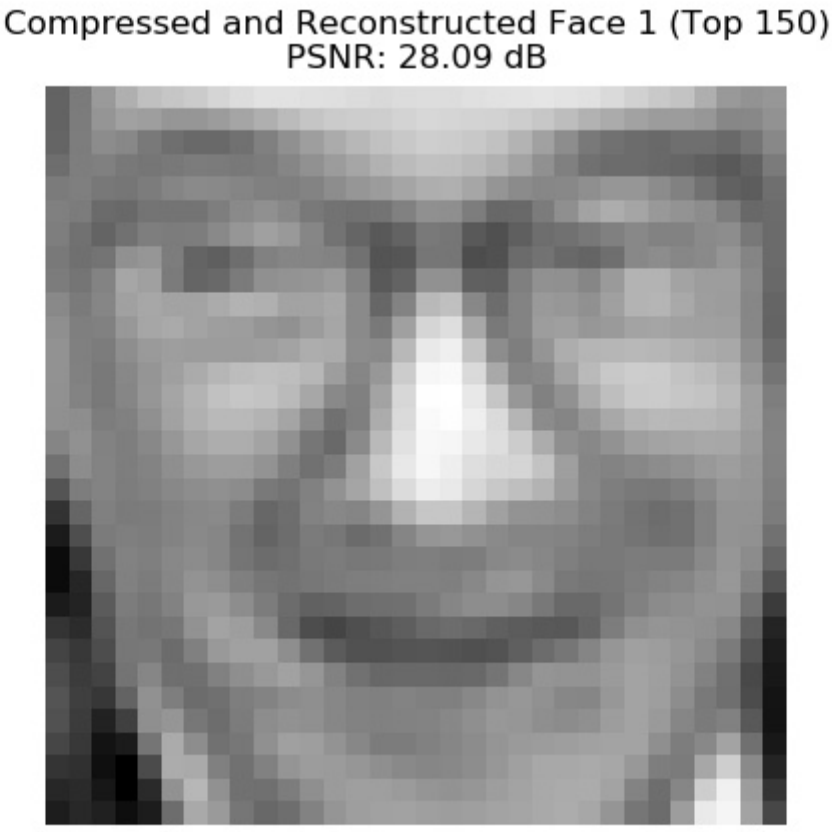
Compressed and Reconstructed Face 3 (Top 100)
PSNR: 27.88 dB

Compressed and Reconstructed Face 4 (Top 100)
PSNR: 31.49 dB


Compressed and Reconstructed Face 5 (Top 100)
PSNR: 29.40 dB

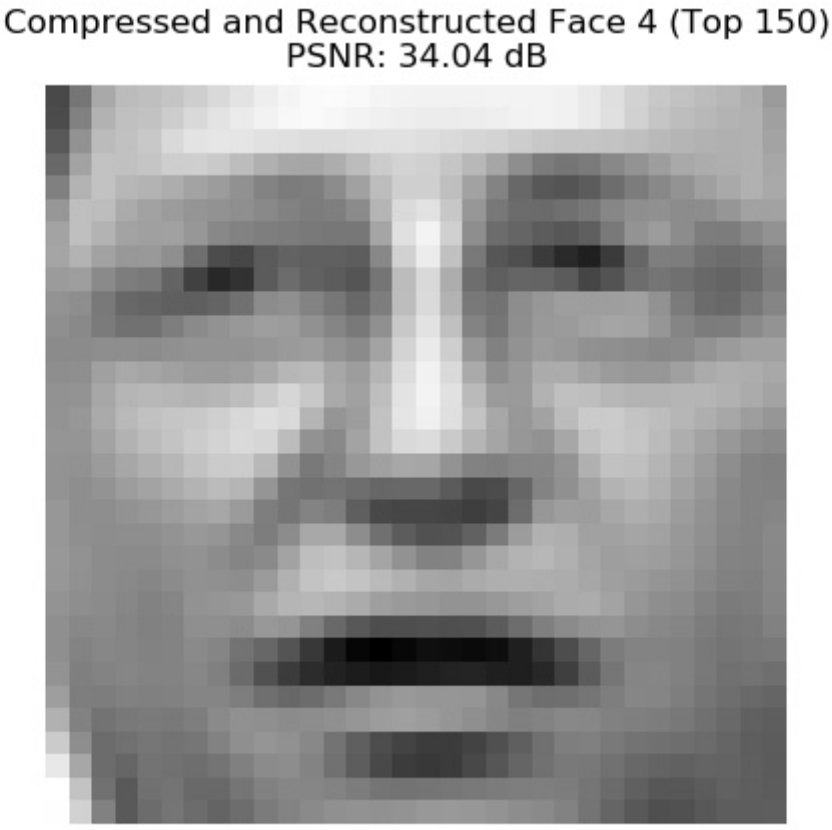
压缩

维度为150：


Compressed and Reconstructed Face 1 (Top 150)
PSNR: 28.09 dB

Compressed and Reconstructed Face 2 (Top 150)
PSNR: 34.59 dB

Compressed and Reconstructed Face 3 (Top 150)
PSNR: 29.75 dB



100

张大图： (按照原图、压缩维度递增的方式展示)





可以很明显看出，随着压缩维度的提高，重建后的图像就和原来的图像越像，也就是还原度较高，这是因为PCA压缩后保留的特征值会越来越多。
当然，从PSNR的计算结果中也能看得出来：

```
PS D:\Course\模式识别\作业二> python -u "d:\Course\模式识别\作业二\main_a.py"
dimension is 10 : 21.299255467933133
dimension is 50 : 25.27046623938486
dimension is 100 : 28.156610449885466
dimension is 150 : 30.349019545990163
PS D:\Course\模式识别\作业二>
```

(b) 利用实现的 PCA 函数，对 lena.png 彩色图像进行压缩和重建。数据位于data/lena.jpg，对该图片分别降维到不同维度(10，50, 100, 150)进行压缩，然后再重建，对比不同维度的压缩和重建效果。将结果保存为results/recovered_lena_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，定量对比要求计算峰值信噪比 PSNR。
首先加载图像：

```
lena_image = Image.open('data/lena.jpg')
lena_data = np.array(lena_image)

plt.imshow(lena_data.astype('uint8'))
plt.axis('off')
plt.title('Original Lena Image')
plt.show()
```

然后对图像进行压缩和重建：这里与灰度图像的处理有所不同，这里提取 RGB 通道数据并分别进行 PCA 压缩和重建，然后再进行合并。

```
compress_dimensions = [10, 50, 100, 150]
for dim in compress_dimensions:
    reconstructed_channels = []
    for channel in range(3):
        channel_data = lena_data[:, :, channel]
        projected_data, components = my_pca(channel_data, dim)
        reconstructed_channel = np.dot(projected_data, components)
        reconstructed_channel = reconstructed_channel + np.mean(channel_data,
axis=0)
        reconstructed_channels.append(reconstructed_channel)

    reconstructed_image = np.stack(reconstructed_channels, axis=-1)

    psnr = calculate_psnr(lena_data, reconstructed_image, channelwise=True)
    print(f'Average PSNR for {dim} components: {psnr:.2f} dB')

    reconstructed_image = Image.fromarray(reconstructed_image.astype('uint8'))
    reconstructed_image.save(f'results/recovered_lena_top_{dim}.jpg')

    plt.imshow(reconstructed_image)
    plt.axis('off')
```

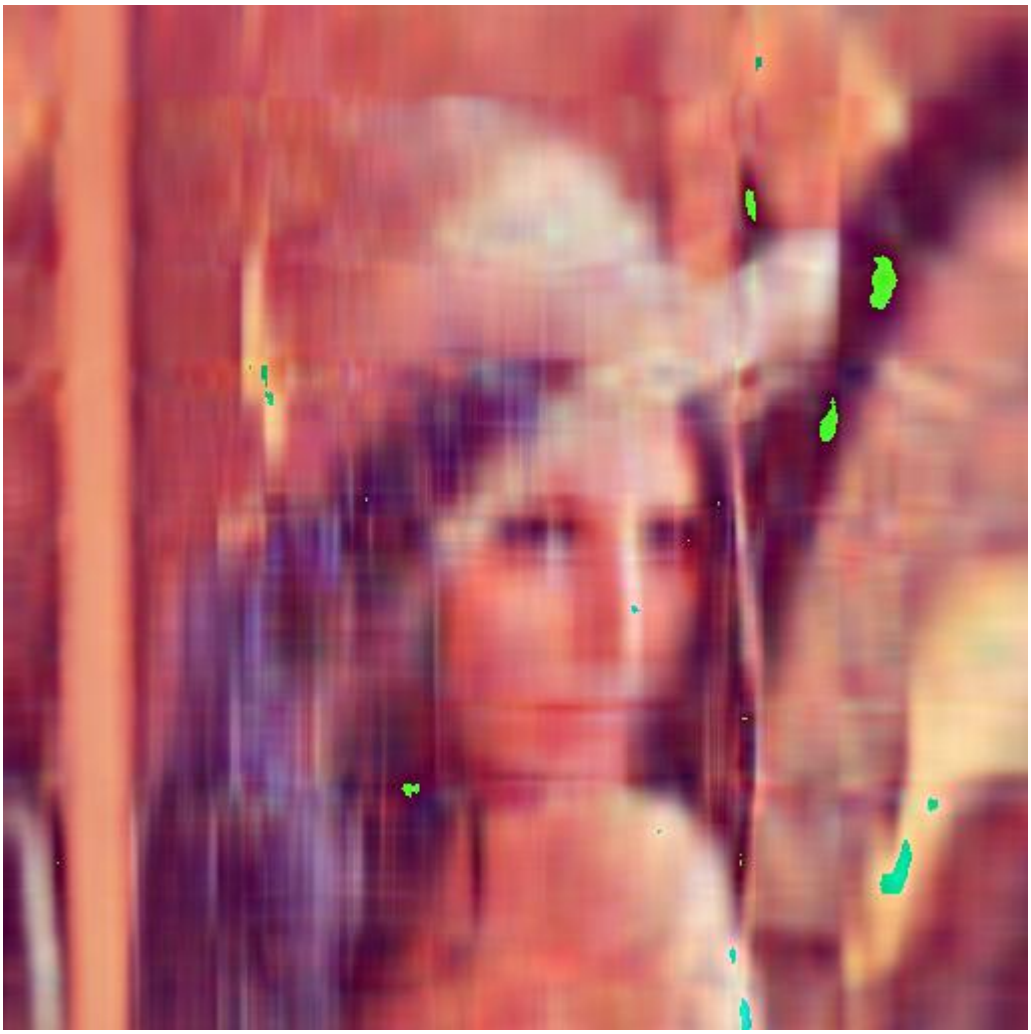
```
    plt.title(f'Reconstructed Lena Image (Top {dim} Components)\nAverage PSNR:
{psnr:.2f} dB')
    plt.show()
```

PSNR的计算方法也和之前灰度图像的有所区别，这里分别计算三个通道的PSNR，再求平均值来代表整个图像的PSNR：

```
def calculate_psnr(original, reconstructed, channelwise=False):
    if channelwise:
        psnrs = []
        for channel in range(original.shape[-1]):
            mse_channel = np.mean((original[:, :, channel] - reconstructed[:, :,
channel]) ** 2)
            max_pixel = 255
            psnr_channel = 10 * np.log10((max_pixel ** 2) / mse_channel)
            psnrs.append(psnr_channel)
        return np.mean(psnrs)
    else:
        mse = np.mean((original - reconstructed) ** 2)
        max_pixel = 255
        psnr = 10 * np.log10((max_pixel ** 2) / mse)
        return psnr
```

结果展示：
原始图像：

压缩维度为10的重建图像：



压缩维度为50的重建图像：

压缩维度为100的重建图像：

压缩维度为150的重建图像：

PSNR计算结果：

```
PS D:\Course\模式识别\作业二> python -u "d:\Course\模式识别\作业二\main_b.py"
Average PSNR for 10 components: 23.08 dB
Average PSNR for 50 components: 30.38 dB
Average PSNR for 100 components: 35.63 dB
Average PSNR for 150 components: 39.85 dB
PS D:\Course\模式识别\作业二>
```

可以看到，对于彩色图像，随着压缩维度的增加，重建后的PSNR值也是越来越大，重建后的图像也是越来越接近原来的图片。

(c) 将 PCA 降维应用到手写数字识别任务上，对图像数据压缩降维，再训练分类模型。任选一种分类方法，可以是作业一中的 Kmeans，也可以是其他准确率更高的方法。观察降维前后，训练速度、准确率等指标是否提升。同时，PCA 也可用于高维数据可视化，将数据维度降到 2 维则可绘制为2 维图像上的数据点，降到 3 维则可绘制为 3 维空间上的数据点。给出MNIST 数据集上分类结果的降维可视化图（2 维或 3 维均可），每个数字类别用不同颜色区分。

实现kmeans函数：

```python
def kmeans(X, k, max_iters, tol):
    def initialize_centers(X, k):
        indices = np.random.choice(X.shape[0], k, replace=False)
        return X[indices]
```

```
    def assign_clusters(X, centers):
        distances = np.sqrt(np.sum((X[:, np.newaxis] - centers) ** 2, axis=2))
        return np.argmin(distances, axis=1)

    def update_centers(X, labels, k):
        centers = np.zeros((k, X.shape[1]))
        for i in range(k):
            centers[i] = np.mean(X[labels == i], axis=0)
        return centers

    centers = initialize_centers(X, k)
    for _ in range(max_iters):
        labels = assign_clusters(X, centers)
        new_centers = update_centers(X, labels, k)
        center_diff = np.sum(np.abs(new_centers - centers))
        if center_diff < tol:
            print(f"Converged after {_} iterations.")
            break
        centers = new_centers
    return centers, labels
```

使用PCA进行降维：

```
n_components = 50
all_images_pca = my_pca(all_images, n_components)
```

结果展示：
不使用PCA降维：

```
PS D:\Course\模式识别\作业一> python -u "d:\Course\模式识别\作业一\main.py"
Converged after 82 iterations. 1e-06
Enter a new label for Cluster 0 (0-9): 5
Enter a new label for Cluster 1 (0-9): 1
Enter a new label for Cluster 2 (0-9): 6
Enter a new label for Cluster 3 (0-9): 3
Enter a new label for Cluster 4 (0-9): 2
Enter a new label for Cluster 5 (0-9): 5
Enter a new label for Cluster 6 (0-9): 0
Enter a new label for Cluster 7 (0-9): 8
Enter a new label for Cluster 8 (0-9): 9
Enter a new label for Cluster 9 (0-9): 7
Cluster 0: 6425 images
Cluster 1: 5275 images
Cluster 2: 5119 images
Cluster 3: 6717 images
Cluster 4: 5307 images
Cluster 5: 8649 images
Cluster 6: 5371 images
Cluster 7: 6714 images
Cluster 8: 10428 images
Cluster 9: 9995 images
Correct count: 41088
Incorrect count: 28912
Accuracy: 58.70%
PS D:\Course\模式识别\作业一>
```

降维10：

```
Correct count: 34026
Incorrect count: 35974
Accuracy: 48.61%
```

降维50：

```
Correct count: 36287
Incorrect count: 33713
Accuracy: 51.84%
```

降维100：

```
Correct count: 37829
Incorrect count: 32171
Accuracy: 54.04%
```

降维150：

```
Correct count: 39354
Incorrect count: 30646
Accuracy: 56.22%
```

方式	准确率
不压缩	58.70%
压缩10	48.61%
压缩50	51.84%

方式	准确率
压缩100	54.04%
压缩150	56.22%

训练速度有很明显的提升，降维后的维度越小速度越快，有时候聚类正确率有了一定的提升。这可能是因为 PCA 去除无用的了噪声与冗余，通过消除特征间的相关性使得距离度量能更好地反映样本间的实际相似度。然而实际上有时候降维前后聚类的正确率似乎没有大的变动，PCA 对于聚集不同数据分布的效果十分有限，而且降维后保留较小的维度时会损失一定的信息导致准确率反而有所下降。