统计学习与模式识别实验一：图像压缩
实验报告

一 实验原理

1.1 PCA 原理概述

介绍

PCA(主成分分析法，Principal Component Analysis)，是一种经典的线性降维方法，广泛用于数据降维、特征变换、图像压缩等。

PCA 的主要思想是将 n 维特征映射到 k 维上，这 k 维是全新的正交特征(即主成分)，是在原有 n 维特征的基础上重新构造出来的 k 维特征。我们有两点期望：这些特征的方向上数据具有最大方差，根据这些特征重构的数据误差最小。可以证明这两点期望是等价的。

PCA 将从原始的空间中顺序地找一组相互正交的坐标轴，新的坐标轴的选择与数据本身是密切相关的。其中，第 1 个新坐标轴选择是原始数据中方差最大的方向，第 2 个新坐标轴选取是与第 1 个坐标轴正交的平面中使得方差最大的，第 3 个轴是与第 1,2 个轴正交的平面中方差最大的。依次类推，可以得到 n 个这样的坐标轴。不过前面小部分坐标轴已经能含有绝大部分方差，即达到降维目的。

推导

这里仅从“最大方差”角度对其进行推导：

设数据为$X = [x_1 x_2 \dots x_N]^T$，具有 N 个样本，D 个特征

首先对数据进行中心化，每个样本点减去均值$x_i - \bar{x}$

投影，求最大方差，考察将$x_n \in \mathbb{R}^D$投影到由$u_1 \in \mathbb{R}^D$定义的一维子空间，得到投影(嵌入)为$(x_i - \bar{x})^\tau \mu_1 \ \ s.t. \mu_1^T \mu_1 = 1$，投影后方差为

$$J = \frac{1}{N}\sum_{i=1}^{N}((x_i - \overline{x})^T \mu_1)^2 = \mu^T \left(\sum_{i=1}^{N} \frac{1}{N}(x_i - \overline{x})(x_i - \overline{x})^T \right) \mu_1 = \mu_1^T S \mu_1$$

该问题归结为一个优化问题：

$$\hat{\mu} = \arg\max \mu_1^T S \mu_1$$

$$s.t. \mu_1^T \mu_1 = 1$$

使用拉格朗日乘数法：

$$L(\mu_1, \lambda) = \mu_1^T S \mu_1 + \lambda(1 - \mu_1^T \mu_1)$$

对 μ_1 求偏导，得到 $\frac{\partial L}{\mu_1} = 2S\mu_1 - 2\lambda\mu_1 = 0$，$S\mu_1 = \lambda\mu_1$。这说明使投影后数据方差最大的方向恰好是最大特征值对应特征向量的方向。以此类推，前 k 个最大特征值对应的特征向量即是我们要寻找的前 k 个投影方向。

计算

传统的方法计算特征值、特征向量时，直接从定义出发，计算特征多项式的根，时间复杂度过高。Python numpy 库的 linalg.eig 采取了优化算法，但导致阶数在 100 以上时不收敛，出现复数。笔者还考虑了使用 python sympy 库，虽然有极高的计算精度，但符号运算时间复杂度极高，甚至不能短时间内处理 1000 阶矩阵的特征值、特征向量求解；以及通过 python 调用 matlab 引擎，虽然没有采用符号计算，且 matlab 计算时能保证精度，但速度很慢，猜测是 python 与 matlab 交互消耗大量时间所致。

事实上对特征值、特征向量的计算可以考虑使用更优的算法，即是通过 SVD 求出 PCA 的主成分(事实上 scipy 库的 PCA 方法正是这样计算的)。这里作一个简单的证明：

PCA 需要对协方差矩阵 C(可以证明其是非负定矩阵)做正交变换，如下：

$$C = \frac{1}{N} A^T A = V \Sigma V^T$$

SVD 得到的有奇异矩阵 V 满足如下关系：

$$A^T A = V \Sigma^2 V^T$$

对比两式，可以发现协方差矩阵的特征向量构成的正交矩阵和 SVD 分解得到的右奇异矩阵其实一样，故其对应的特征(奇异)向量是一样的；而奇异值正好是特征值的平方。

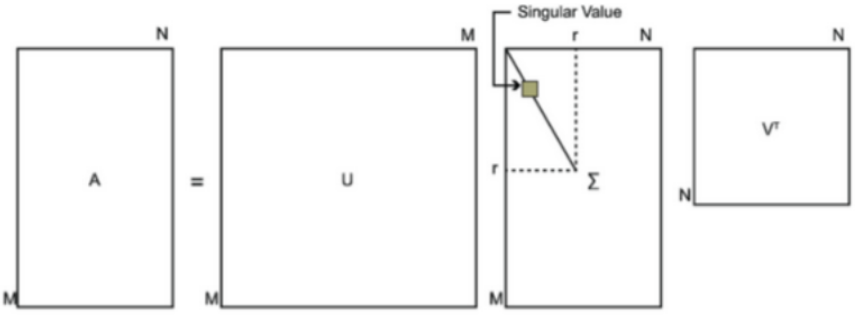
1.2 SVD 原理概述

介绍

SVD (奇异值分解，Singular Value Decomposition)是在机器学习领域广泛应用的算法，它不光可以用于降维算法中的特征分解，还可以用于推荐系统，以及自然语言处理等领域，是很多机器学习算法的基石。

推导

SVD 也是对矩阵进行分解，但是和特征分解不同，SVD 并不要求要分解的矩阵为方阵。假设我们的矩阵 A 是一个 $m \times n$ 的矩阵，那么我们定义矩阵 A 的 SVD 为 $A=U\sum V^T$。其中 U、V 都是正交矩阵，$\sum$ 是一个 $m \times n$ 的矩阵，除主对角元上的元素以外全是 0。四个矩阵的形状如图。



由 $A=U\sum V^T \Rightarrow A^TA=V\sum^2 V^T$，可以求得右奇异矩阵，同理可求得左奇异矩阵，再由 $A=U\sum V^T \Rightarrow AV=U\sum \Rightarrow Av_i=\sigma_i u_i \Rightarrow \sigma_i = Av_i / u_i$ 进而求得奇异值矩阵。

二 代码实现

2.1 基于 PCA 的人脸识别

导入必要的库，作一些设置

```
import numpy as np
from numpy import linalg
import os
from PIL import Image
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from random import *
sns.set_style("dark")
plt.rc('font',family='Times New Roman')
```

创建 PCA class，初始化

```
class PCA():

    def __init__(self):
        self.images=[]
```

定义函数，从文件夹中读入数据

```
def readdata(self):
    path = './ORL'
    self.size = set()
    for file in os.listdir(path):
        for item in os.listdir(path + '/' + file):
            im = Image.open(path + '/' + file + '/' + item)
            self.images.append(list(im.getdata()))
            self.size.add(im.size)
    if len(self.size)==1:
        self.size=list(self.size)[0]
        self.images=np.array(self.images,dtype=float)
        print('image size:',self.size)
    else:raise Exception("size not match")
    N, D = len(self.images), len(self.images[0])  # 样本数，特征数
    self.D=D;self.N=N
```

计算主成分，计算数据的变换

```
def pca(self):
    X=np.array(self.train.copy())
    D=self.D;N=self.N
    print(f'sample number:{N},feature number:{D}')
    X = np.array(X)
    u,sigma,eigvec=linalg.svd(X)
    self.eigval=[i**0.5 for i in sigma]
    self.eigvec=eigvec
    print(f'eigvec number:{len(eigvec)},eigvec
dim:{len(eigvec[0])}' )
```

```
    Zall=np.dot(eigvec,X.transpose())
    self.Zall=Zall
```

从变换后的数据还原原数据，并绘制图像进行观察

```
def recompose(self,idx):
    X=np.array(self.images).transpose()
    fig=plt.figure(figsize=(8,7))
    ax=fig.add_subplot(5,6,1)
    ori=self.shape(X[:,idx])
    ax.imshow(ori,cmap = plt.cm.gray)
    plt.title('Original', y=-0.25)
    plt.xticks([]);plt.yticks([])
    # ax.set_title(r'Original')
    i=1
    for k in
[1,2,3,4,5,7,10,15,20,50,100,200,300,400,500,600,700,800,900,1000,200
0,3000,4000,5000,6000,7000,8000,9000,10304]:
        print(k)
        i+=1
        ax = fig.add_subplot(5, 6, i)
        self.embedding(k)
        Z = self.Z;U = self.U
        X_ = np.dot(U,Z)
        img = self.shape(X_[:, idx])
        ax.imshow(img,cmap = plt.cm.gray)
        plt.title(r'k={}'.format(k),y=-0.25)
        plt.xticks([]);plt.yticks([])
    plt.subplots_adjust(left=None, bottom=None, right=None, top=None,
wspace=0, hspace=0.3)
    # plt.show()
    plt.savefig('./images/'+str(idx)+'.png',dpi=2000)
```

通过主成分来绘制特征脸

```
def featureface(self):
    fig=plt.figure(figsize=(8,7))
    features=self.eigvec
    for i in range(1,49+1):
        ax=fig.add_subplot(7,7,i)
        ff=self.shape(features[i])
        ax.imshow(ff, cmap=plt.cm.gray)
        plt.title('{}'.format(i), y=-0.4)
        plt.xticks([]);plt.yticks([])
    plt.subplots_adjust(left=None, bottom=None, right=None, top=None,
wspace=0, hspace=0.5)
    plt.savefig('./images/featureface' + '.png', dpi=2000)
```

将数据变换为固定的形状

```
def shape(self,image):
    return image.reshape(self.size[1],self.size[0])
```

计算使用不同数量主成分时的误差

```
def loss(self,A,B):
    absloss=sum([(A[i]-B[i])**2 for i in range(len(A))])
    s=sum([Bi**2 for Bi in B])
    relaloss=absloss/s
    return absloss,relaloss

def lossplot(self):
    X=self.images
    absloss=[]
    relaloss=[]
    n=10304
    self.embedding(n)
    Zall = self.Zall
    eigvec = self.eigvec.transpose()
    for k in range(1,n):
        print(k)
        X_ = np.dot(eigvec[:,:k], Zall[:k,0:1])
        m=1
        al=0;rl=0
        for i in range(1):
            dal,drl=self.loss(X[i],[i[0] for i in X_[:]])
            al+=dal;rl+=drl
        absloss.append([al/m,k]);relaloss.append([rl/m,k])
    absloss = pd.DataFrame(absloss, columns=['Absolute error ', 'k'])
    relaloss = pd.DataFrame(relaloss, columns=['Relative error ',
'k'])
    fig=plt.figure(figsize=(8,3.5))
    ax=fig.add_subplot(1,2,1)
    sns.lineplot(data=absloss,y='Absolute error ',x='k')
    ax=fig.add_subplot(1,2,2)
    sns.lineplot(data=relaloss,y='Relative error ',x='k')
    plt.subplots_adjust(left=None, bottom=None, right=None, top=None,
wspace=0.2, hspace=0.5)
    plt.savefig('./images/loss' + '.png', dpi=2000)
```

在数据集中找到和目标最像的脸

```
def findmostsimilar(self,train,test,t):
    n=len(train)
    print('len train:',n)
    dis=[]
    for i in range(n):
        traini=train[i]
        dis.append([sum([(traini[j]-test[j])**2 for j in
range(400)]),i])
    dis.sort()
    dis=dis[:3]
    idx=[d[1] for d in dis]
    fig=plt.figure(figsize=(8,4))
    ax=fig.add_subplot(2,4,1)
    ax.imshow(self.shape(self.images[self.testsid[t]]),
cmap=plt.cm.gray)
    plt.title('original', y=-0.25)
    plt.xticks([]);plt.yticks([])
    sim=['/most similar','/second similar','/third similar']
    for i in range(1,4):
        ax = fig.add_subplot(2, 4, i+1)
        ax.imshow(self.shape(self.images[self.trainid[idx[i-1]]]),
cmap=plt.cm.gray)
        plt.title('original'+sim[i-1], y=-0.25)
        plt.xticks([]);plt.yticks([])
    ax=fig.add_subplot(2,4,5)
    ax.imshow(self.shape(self.recompose2(test)), cmap=plt.cm.gray)
    plt.title('k=400', y=-0.25)
    plt.xticks([]);plt.yticks([])
    for i in range(5,8):
        ax = fig.add_subplot(2, 4, i+1)
        ax.imshow(self.shape(self.recompose2(train[idx[i-5]])),
cmap=plt.cm.gray)
        plt.title('k=400'+sim[i-5], y=-0.25)
        plt.xticks([]);plt.yticks([])
    plt.subplots_adjust(left=None, bottom=None, right=None, top=None,
wspace=0.2, hspace=0.5)
    plt.savefig('./images/similar'+str(t) + '.png', dpi=2000)
```

计算平均脸

```
def averageface(self):
    n=len(self.images)
    ave=np.array([sum([self.images[j][i] for j in range(n)])/n for i
in range(len(self.images[0]))])
```

```
    plt.imshow(self.shape(ave), cmap=plt.cm.gray)
    plt.savefig('./images/average')
```

考察降维效果，展示特征脸

```
def test(self):
    self.readdata()
    self.train=self.images
    self.pca()
```

利用降维后的数据作人脸识别

```
def test2(self):
    self.readdata()
    n=len(self.images)
    items=[i for i in range(n)]
    shuffle(items)
    trainnum=n//10*9
    print(f'train number:{trainnum},test number:{n-trainnum}')
    train=items[:trainnum]
    tests=items[trainnum:]
    self.train=self.images
    self.trainid=train;self.testsid=tests
    testsitem=np.array([self.images[i] for i in tests])
    # self.images=[self.images[i] for i in train]
    self.pca()
    eigvec=self.eigvec
    Ztrain=np.dot(eigvec[:400],np.array([self.images[i] for i in
train]).transpose()).transpose()
    print(len(Ztrain),len(Ztrain[0]))
    Ztest=np.dot(eigvec[:400],testsitem.transpose()).transpose()
    print(len(Ztest),len(Ztest[0]))
    for i in range(len(Ztest)):
        print(i)
        self.findmostsimilar(Ztrain,Ztest[i],i)
```

显示平均脸

```
def test3(self):
    self.readdata()
    self.averageface()
```

主函数，调用不同的测试函数，作不同类型的测试

```
def main():
    test=PCA()
    # test.test()
    # test.test2()
```

```
    test.test3()

if __name__=='__main__':
    main()
```

2.2 基于 PCA 的图像压缩

导入库、设置和创建 PCA class 的代码省略。

定义函数，从文件夹中读入数据

```
def readdata(self,show=False):
    path = './images'
    file='/butterfly.bmp'
    im=Image.open(path+file)
    self.images=list(im.getdata())
    self.size=im.size
    # print(self.images)
    self.R,self.G,self.B=map(np.array,list(zip(*self.images)))
    if
show==True:self.showdata(self.R,self.G,self.B,'./images/figshow2.png'
)
```

展示图片，将三个色道分别和一起的图像绘制在一张图中并展示出来。两种类型的数据均可被绘制出来。

```
def showdata(self,R,G,B,path,flag=0):
    # print('R G B')
    R,G,B=map(self.regular,[R,G,B])
    # print(R);print(G);print(B)
    if flag==0:
        fig=plt.figure(figsize=(7,5))
        ax=fig.add_subplot(2,2,1)
        plt.imshow(self.shape2(list(zip(R,G,B))))
        plt.title('Original',y=-0.4)
        ax=fig.add_subplot(2,2,2)
        plt.imshow(self.shape(R), cmap='Reds')
        plt.title('R',y=-0.4)
        ax=fig.add_subplot(2,2,3)
        plt.imshow(self.shape(G), cmap='Greens')
        plt.title('G', y=-0.4)
        ax=fig.add_subplot(2,2,4)
        plt.imshow(self.shape(B), cmap='Blues')
        plt.title('B', y=-0.4)
        plt.subplots_adjust(left=None, bottom=None, right=None,
top=None, wspace=0.3, hspace=0.3)
```

```
        plt.show()
        # plt.savefig(path,dpi=2000, bbox_inches='tight')
    else:
        fig=plt.figure(figsize=(7,5))
        ax=fig.add_subplot(2,2,1)
        plt.imshow([[(int(R[j][i]),int(G[j][i]),int(B[j][i])) for j in
range(len(R))] for i in range(len(R[0]))])
        plt.title('RGB',y=-0.4)
        ax=fig.add_subplot(2,2,2)
        plt.imshow(R.transpose(), cmap='Reds')
        plt.title('R',y=-0.4)
        ax=fig.add_subplot(2,2,3)
        plt.imshow(G.transpose(), cmap='Greens')
        plt.title('G', y=-0.4)
        ax=fig.add_subplot(2,2,4)
        plt.imshow(B.transpose(), cmap='Blues')
        plt.title('B', y=-0.4)
        plt.subplots_adjust(left=None, bottom=None, right=None,
top=None, wspace=0.3, hspace=0.3)
        plt.show()
        # plt.savefig(path,dpi=2000, bbox_inches='tight')
```

将像素值线性变换到[0,255]区间，便于绘图。

```
def regular(self,RGB):
    if type(RGB[0])!=np.ndarray and type(RGB[0])!=list:
        m=min(RGB);M=max(RGB)
        return [int((rgb-m)/(M-m)*255) for rgb in RGB]
    else:
        m,M=1e100,-1e100
        for i in RGB:
            for j in i:m=min(m,j);M=max(M,j)
        for i in range(len(RGB)):
            for j in range(len(RGB[0])):RGB[i][j]=int((RGB[i][j]-m)/(M-
m)*255)
        return RGB
```

计算主成分，计算数据的变换。这里提供了两种方法，直接使用 linalg.eig 和使用 linalg.svd 来求得主成分。

```
def pca(self,pcamethod='singular'):
    self.R, self.G, self.B = map(np.array, list(zip(*self.images)))
    D=self.size[1];N=self.size[0]
    print(f'sample number:{N},feature number:{D}')
    if pcamethod == 'eig': R, G, B = map(self.pca_1, map(self.shape2,
[self.R, self.G, self.B]))
    else: R, G, B = map(self.pca_2, map(self.shape2, [self.R, self.G,
```

```
self.B]))
    self.egivecR, self.ZR = R
    self.egivecG, self.ZG = G
    self.egivecB, self.ZB = B

def pca_1(self,X):
    X=np.array(X)
    D = self.size[1];N = self.size[0]
    cov=np.dot(X.transpose(),X)
    eigval,eigvec=np.linalg.eig(cov)
    eigval=np.real(eigval)
    eigvec=[list(i) for i in np.real(eigvec)]
    vv=list(zip(eigval,eigvec))
    vv.sort(reverse=True)
    eigvec=np.array(list(zip(*vv))[1])
    print(f'eigvec number:{len(eigvec)},eigvec dim:{len(eigvec[0])}')
    Zall = np.dot(eigvec, X.transpose())
    return eigvec,Zall

def pca_2(self,X):
    X=np.array(X)
    D = self.size[1];N = self.size[0]
    u, sigma, eigvec = linalg.svd(X)
    eigval = [i ** 0.5 for i in sigma]
    print(f'eigvec number:{len(eigvec)},eigvec dim:{len(eigvec[0])}')
    Zall = np.dot(eigvec, X.transpose())
    return eigvec,Zall
```

通过主成分来绘制特征。

```
def featureface(self):
    fig=plt.figure(figsize=(8,7))
    eigs=[self.egivecR,self.egivecG,self.egivecB]
    Zs=[self.ZR,self.ZG,self.ZB]
    for k in range(1,49+1):
        for i in range(3):
            if i == 0:R = np.array([j*eigs[i][k,:] for j in
Zs[i][k,:]]).transpose()
            elif i == 1:G = np.array([j*eigs[i][k,:] for j in
Zs[i][k,:]]).transpose()
            else:B = np.array([j*eigs[i][k,:] for j in
Zs[i][k,:]]).transpose()
        ax=fig.add_subplot(7,7,k)
        R, G, B = map(self.regular, [R, G, B])
        plt.imshow([[(int(R[j][i]),int(G[j][i]),int(B[j][i])) for j in
range(len(R))] for i in range(len(R[0]))])
```

```
        plt.title('{}'.format(k), y=-0.45)
        plt.xticks([]);plt.yticks([])
    print('over')
    plt.subplots_adjust(left=None, bottom=None, right=None, top=None,
wspace=0, hspace=0.3)
    plt.savefig('./images/featureface' + '.png',dpi=2000,
bbox_inches='tight')
```

计算使用不同数量主成分时的误差。

```
def loss(self,A,B):
    l=lambda x,y:sum([(x[i]-y[i])**2 for i in range(len(x))])
    abl=0
    normb=0
    for i in range(len(A)):
        for j in range(len(A[0])):
            abl+=l(A[i][j],B[i][j])
            normb+=sum([s**2 for s in B[i][j]])
    if normb==0:raise Exception('b is zero')
    return abl,abl/normb

def lossplot(self):
    X=self.images
    absloss=[]
    relaloss=[]
    X=self.shape2(X)
    n=len(self.egivecR)
    # n=3
    eigs = [self.egivecR, self.egivecG, self.egivecB]
    Zs = [self.ZR, self.ZG, self.ZB]
    for k in range(1,n):
        print(k)
        for i in range(3):
            if i==0:R=np.dot(eigs[i][:k].transpose(),Zs[i][:k])
            elif i==1:G=np.dot(eigs[i][:k].transpose(),Zs[i][:k])
            else:B=np.dot(eigs[i][:k].transpose(),Zs[i][:k])
        X_ = [[(R[j][i],G[j][i],B[j][i]) for j in range(len(R))] for i
in range(len(R[0]))]
        abl,rel=self.loss(X_,X)
        absloss.append([abl,k]);relaloss.append([rel,k])
    absloss = pd.DataFrame(absloss, columns=['Absolute error ', 'k'])
    relaloss = pd.DataFrame(relaloss, columns=['Relative error ',
'k'])
    fig=plt.figure(figsize=(8,3.5))
    ax=fig.add_subplot(1,2,1)
    sns.lineplot(data=absloss,y='Absolute error ',x='k')
```

```
    ax=fig.add_subplot(1,2,2)
    sns.lineplot(data=relaloss,y='Relative error ',x='k')
    plt.subplots_adjust(left=None, bottom=None, right=None, top=None,
wspace=0.2, hspace=0.5)
    plt.savefig('./images/loss' + '.png', dpi=2000)
```

考察不同数量主成分的效果。

```
def test(self):
    self.readdata()
    k1=[1,2,3,4,5,7,10,20,30,40,50,60,70,80,90,100,120,150,243]
    k2=[1,2,3,4,5,7,10,20,30,50,100,150,200,250,300,350,437]
    self.pca('singular')
    # # raise Exception('stop')
    eigs=[self.egivecR,self.egivecG,self.egivecB]
    Zs=[self.ZR,self.ZG,self.ZB]
    for k in k2:
        print(k)
        for i in range(3):
            if i==0:R=np.dot(eigs[i][:k].transpose(),Zs[i][:k])
            elif i==1:G=np.dot(eigs[i][:k].transpose(),Zs[i][:k])
            else:B=np.dot(eigs[i][:k].transpose(),Zs[i][:k])
        print(len(R),len(R[0]))
        self.showdata(R,G,B,'./images/axis1_'+str(k)+'.png',flag=1)
```

测试函数分别调用了测试主成分、绘制特征图、绘制误差曲线的函数，此处代码省略。

主函数，调用不同的测试函数，作不同类型的测试。

```
def main():
    test=PCA()
    test.test()
    # test.test2()
    # test.test3()

if __name__=='__main__':
    main()
```

2.3 基于 SVD 的图像压缩

SVD 的代码框架与 PCA 几乎完全相同，仅在求主成分、求

U/sigma/V 的地方，求压缩后的图像时有所不同，因此在此一并贴出，而不对每一部分作详细解释。

```
import numpy as np
from numpy import linalg
import os
from PIL import Image
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from random import *
sns.set_style("dark")

plt.rc('font',family='Times New Roman')

class SVD():

    def __init__(self):
        self.images=[]

    def readdata(self,show=False):
        path = './images'
        file='/butterfly.bmp'
        im=Image.open(path+file)
        self.images=list(im.getdata())
        self.size=im.size
        # print(self.images)
        self.R,self.G,self.B=map(np.array,list(zip(*self.images)))
        if
show==True:self.showdata(self.R,self.G,self.B,'./images/figshow2.png'
)

    def showdata(self,R,G,B,path,flag=0):
        # print('R G B')
        R,G,B=map(self.regular,[R,G,B])
        # print(R);print(G);print(B)
```

```
        if flag==0:
            fig=plt.figure(figsize=(7,5))
            ax=fig.add_subplot(2,2,1)
            plt.imshow(self.shape2(list(zip(R,G,B))))
            plt.title('Original',y=-0.4)
            ax=fig.add_subplot(2,2,2)
            plt.imshow(self.shape(R), cmap='Reds')
            plt.title('R',y=-0.4)
            ax=fig.add_subplot(2,2,3)
            plt.imshow(self.shape(G), cmap='Greens')
            plt.title('G', y=-0.4)
            ax=fig.add_subplot(2,2,4)
            plt.imshow(self.shape(B), cmap='Blues')
            plt.title('B', y=-0.4)
            plt.subplots_adjust(left=None, bottom=None, right=None,
top=None, wspace=0.3, hspace=0.3)
            # plt.show()
            plt.savefig(path,dpi=2000, bbox_inches='tight')
        else:
            fig=plt.figure(figsize=(7,5))
            ax=fig.add_subplot(2,2,1)
            plt.imshow([[(int(R[j][i]),int(G[j][i]),int(B[j][i])) for j
in range(len(R))] for i in range(len(R[0]))])
            plt.title('RGB',y=-0.4)
            ax=fig.add_subplot(2,2,2)
            plt.imshow(R.transpose(), cmap='Reds')
            plt.title('R',y=-0.4)
            ax=fig.add_subplot(2,2,3)
            plt.imshow(G.transpose(), cmap='Greens')
            plt.title('G', y=-0.4)
            ax=fig.add_subplot(2,2,4)
            plt.imshow(B.transpose(), cmap='Blues')
            plt.title('B', y=-0.4)
            plt.subplots_adjust(left=None, bottom=None, right=None,
top=None, wspace=0.3, hspace=0.3)
            # plt.show()
```

```
            plt.savefig(path,dpi=2000, bbox_inches='tight')


    def regular(self,RGB):
        if type(RGB[0])!=np.ndarray and type(RGB[0])!=list:
            m=min(RGB);M=max(RGB)
            return [int((rgb-m)/(M-m)*255) for rgb in RGB]
        else:
            m,M=1e100,-1e100
            for i in RGB:
                for j in i:m=min(m,j);M=max(M,j)
            for i in range(len(RGB)):
                for j in range(len(RGB[0])):RGB[i][j]=int((RGB[i][j]-
m)/(M-m)*255)
            return RGB


    def svd(self):
        self.R, self.G, self.B = map(np.array,
list(zip(*self.images)))
        D=self.size[1];N=self.size[0]
        print(f'sample number:{N},feature number:{D}')
        R, G, B = map(self.svd_1, map(self.shape2, [self.R, self.G,
self.B]))
        self.UR, self.sigmaR, self.VR = R
        self.UG, self.sigmaG, self.VG = G
        self.UB, self.sigmaB, self.VB = B


    def svd_1(self,X):
        X=np.array(X)
        D = self.size[1];N = self.size[0]
        U, sigma_, V = linalg.svd(X)


        print(f'eigvec number:{len(V)},eigvec dim:{len(V[0])}')
        n=max(len(U),len(V))
        sigma=[[sigma_[i] if i==j and i<len(sigma_) else 0 for i in
range(n)] for j in range(n)]
        return U,sigma,V
```

```
    def recompose(self,eigvec,cor):
        k=len(cor)
        return np.dot(eigvec[:k].transpose(),cor)

    def featureface(self):
        fig=plt.figure(figsize=(8,7))
        Us = [self.UR, self.UG, self.UB]
        sigmas = [self.sigmaR, self.sigmaG, self.sigmaB]
        Vs = [self.VR, self.VG, self.VB]
        for k in range(1,49+1):
            for i in range(3):
                if i == 0: R = np.dot(np.array([j*Us[i][:, k] for j in
sigmas[i][k]]).transpose(), Vs[i][:])
                elif i == 1:G = np.dot(np.array([j*Us[i][:, k] for j in
sigmas[i][k]]).transpose(), Vs[i][:])
                else:B = np.dot(np.array([j*Us[i][:, k] for j in
sigmas[i][k]]).transpose(), Vs[i][:])
                ax=fig.add_subplot(7,7,k)
            R, G, B = map(self.regular, [R, G, B])
            plt.imshow([[(int(R[i][j]),int(G[i][j]),int(B[i][j])) for j
in range(len(R[0]))] for i in range(len(R))])
            plt.title('{}'.format(k), y=-0.45)
            plt.xticks([]);plt.yticks([])
        print('over')
        plt.subplots_adjust(left=None, bottom=None, right=None,
top=None, wspace=0, hspace=0.3)
        plt.savefig('./images/featureface' + '.png',dpi=2000,
bbox_inches='tight')

    # 将向量变为矩阵
    def shape(self,image):
        return image.reshape(self.size[1],self.size[0])

    # 将元素为元组的向量转换为矩阵
    def shape2(self,image):
```

```
    imnew = []
    line = []
    s=self.size
    for i in range(len(image) + 1):
        if i % s[0] == 0 and i != 0:
            imnew.append(line)
            line = []
        if i != len(image): line.append(image[i])
    return imnew


def loss(self,A,B):
    l=lambda x,y:sum([(x[i]-y[i])**2 for i in range(len(x))])
    abl=0
    normb=0
    for i in range(len(A)):
        for j in range(len(A[0])):
            abl+=l(A[i][j],B[i][j])
            normb+=sum([s**2 for s in B[i][j]])
    if normb==0:raise Exception('b is zero')
    return abl,abl/normb

def lossplot(self):
    X=self.images
    absloss=[]
    relaloss=[]
    X=self.shape2(X)
    # n=len(self.VB)
    n=len(self.UR)
    Us= [self.UR, self.UG, self.UB]
    sigmas = [self.sigmaR, self.sigmaG, self.sigmaB]
    Vs = [self.VR, self.VG, self.VB]
    for k in range(1,n):
        print(k)
        for i in range(3):
            if
```

```
i==0:R=np.dot(np.dot(Us[i][:,:k],sigmas[i][:k]),Vs[i][:])
                elif
i==1:G=np.dot(np.dot(Us[i][:,:k],sigmas[i][:k]),Vs[i][:])

else:B=np.dot(np.dot(Us[i][:,:k],sigmas[i][:k]),Vs[i][:])
            X_ = [[(R[i][j],G[i][j],B[i][j]) for j in range(len(R[0]))]
for i in range(len(R))]
            abl,rel=self.loss(X_,X)
            absloss.append([abl,k]);relaloss.append([rel,k])
        absloss = pd.DataFrame(absloss, columns=['Absolute error ',
'k'])
        relaloss = pd.DataFrame(relaloss, columns=['Relative error ',
'k'])
        fig=plt.figure(figsize=(8,3.5))
        ax=fig.add_subplot(1,2,1)
        sns.lineplot(data=absloss,y='Absolute error ',x='k')
        ax=fig.add_subplot(1,2,2)
        sns.lineplot(data=relaloss,y='Relative error ',x='k')
        plt.subplots_adjust(left=None, bottom=None, right=None,
top=None, wspace=0.2, hspace=0.5)
        plt.savefig('./images/loss' + '.png', dpi=2000,
bbox_inches='tight')
        # plt.show()

    # 考察不同数量主成分的效果
    def test(self):
        self.readdata()
        ks=[1,2,3,4,5,7,10,20,30,50,100,150,200,250,300,350,437]
        self.svd()
        Us= [self.UR, self.UG, self.UB]
        sigmas = [self.sigmaR, self.sigmaG, self.sigmaB]
        Vs = [self.VR, self.VG, self.VB]
        for k in ks:
            print(k)
            for i in range(3):
                if
```

```
i==0:R=np.dot(np.dot(Us[i][:,:k],sigmas[i][:k]),Vs[i][:])
                elif
i==1:G=np.dot(np.dot(Us[i][:,:k],sigmas[i][:k]),Vs[i][:])

else:B=np.dot(np.dot(Us[i][:,:k],sigmas[i][:k]),Vs[i][:])
            print(len(R),len(R[0]))

self.showdata(R.transpose(),G.transpose(),B.transpose(),'./images/axi
s1_'+str(k)+'.png',flag=1)

    # 特征
    def test2(self):
        self.readdata()
        self.svd()
        self.featureface()

    # loss
    def test3(self):
        self.readdata()
        self.svd()
        self.lossplot()

def main():
    test=SVD()
    # test.test()
    # test.test2()
    test.test3()

if __name__=='__main__':
    main()
```

三个实验全部代码详见附件。

三 结果讨论及展示

3.1 基于 PCA 的人脸识别

实验使用 ORL 数据库，该数据库包括 40 个不同人，每人 10 幅图像，共 400 幅。 每幅原始图像为 256 个灰度级，分辨率为 112×92。笔者用该数据集完成了数据降维(从 112×92 降为 400 维或其他维度)，并作了人脸识别。

将所有图片的所有点像素值取平均，并重绘，可以得到下图，数据集的“平均脸”。

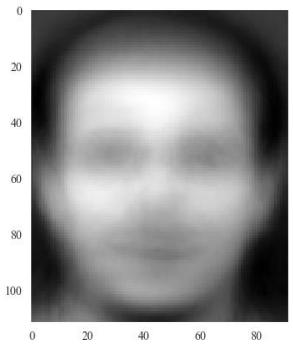


图 1 平均脸

每张图都是一个 112×92 的矩阵，将其变成一维向量，则该向量有 112×92=10304 维。同时数据集有 400 张图片，即数据集的大小是 400×10304，400 个样本，10304 个特征。使用 PCA 方法，可以重构 10304 个主成分，展示前 49 个主成分（“特征脸”）如下。

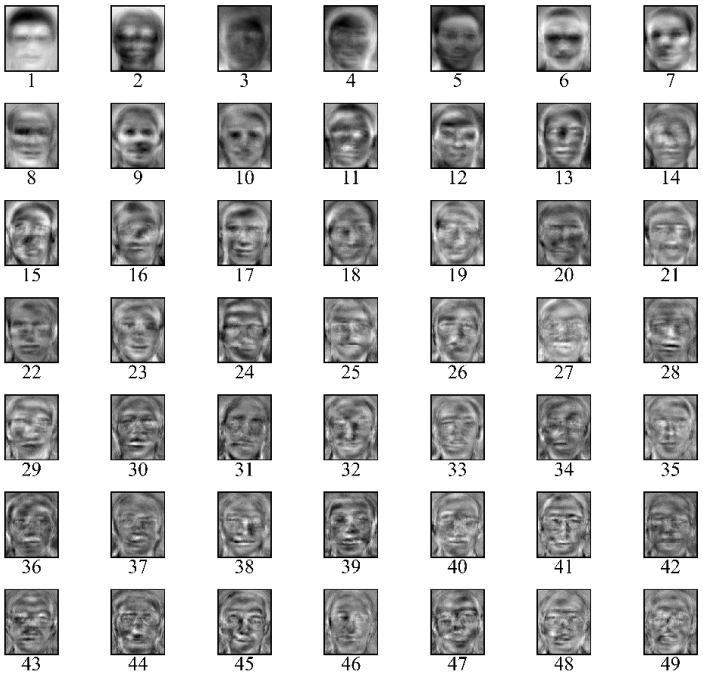
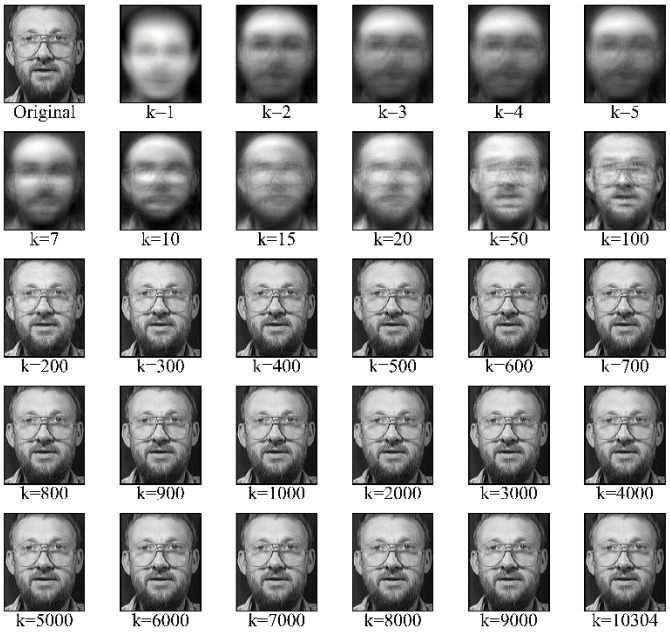


图 2 特征脸

选取不同数量的主成分对图像进行重构，选择两张图片展示，效果如下。可见，在 k=50 或 100 时已经能大致看出人物，而 k=300 时已经能完全辨认出人物，k=400 时已经看不出与原图的差异。400 相比 10304 是一个很小的值，不到 4%的主成分能达到如此好的效果确实令人感到惊奇。这也为后续人脸识别时选取 k=400 提供了支撑。

(a)



(b)

图 3 人脸图片的重构

我们以第一张图片为例，计算绝对误差与相对误差如图 4 所示，这与我们在图 3 中得到的结论是一致的，误差随选取主成分的数量增加而迅速下降。

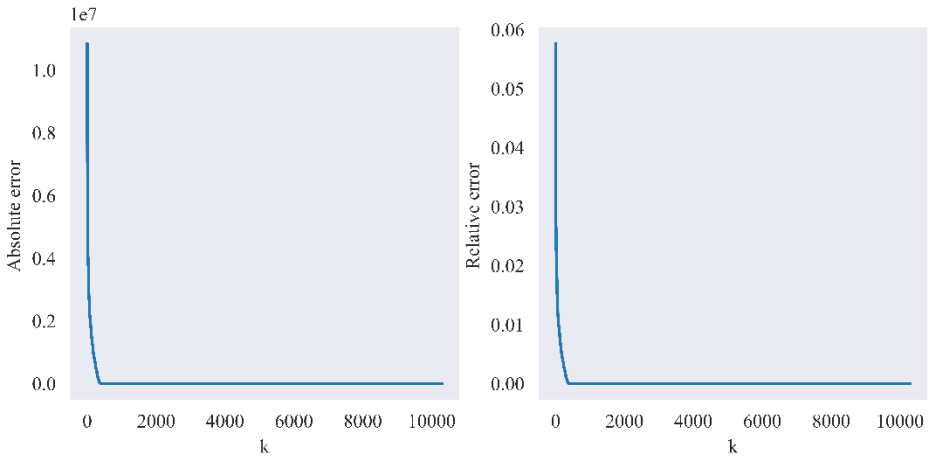
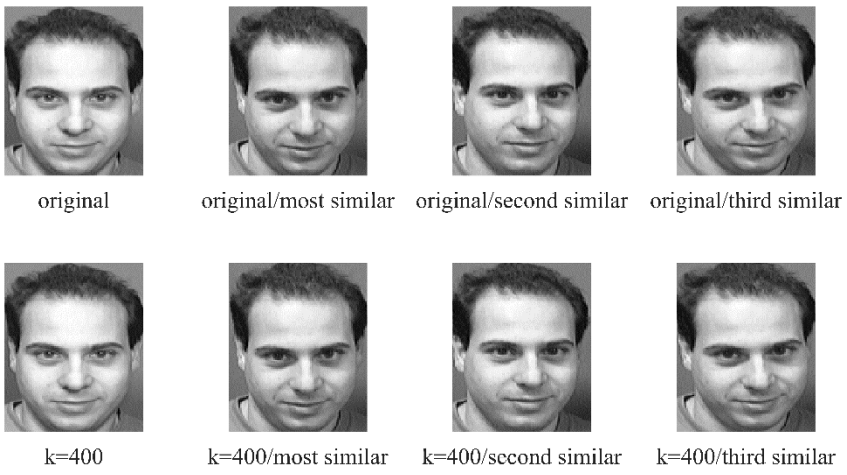


图 4 误差随选取主成分数量的变化

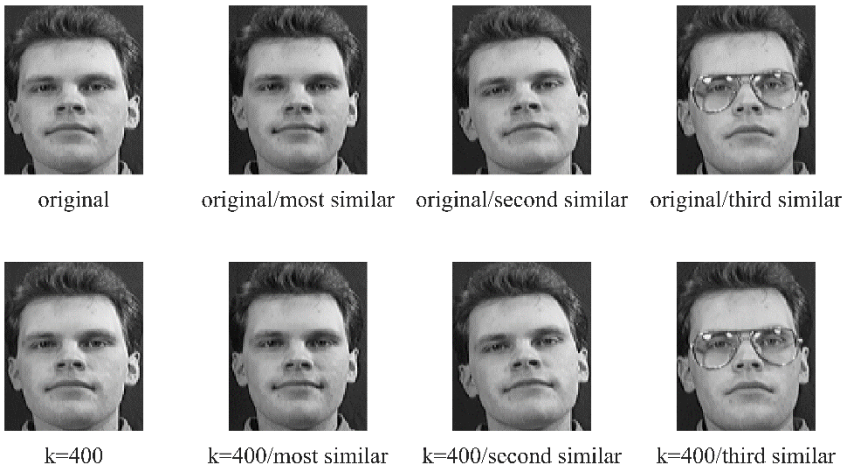
此处人脸识别的目的是在测试集中选取一张图片，在数据集(与测试集无交集)中找到一张与之最相似的图片，若为同一个人则视为成功。

事实上直接计算 10304 维向量之间的距离，选取距离最近的向量即可，但维数高计算复杂度也很高，故我们降为 400 维再作操作。以 3 张图片为例展示，效果如下。

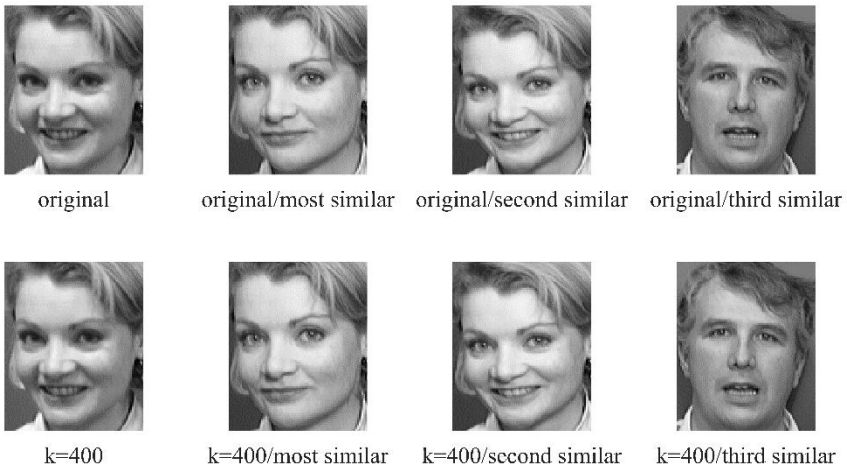
可见三张图片找到的最相似图片都满足要求。(a),(b)中前三名最相似图片与原始图片均为同一个人，值得一提的是(b)中最后一张图的人戴上了眼镜却依旧被选择了出来，说明了算法找到了主要特征而忽略了次要特征。(c)中第三相似的图片与原图不一致，可以作这样的推测：程序中，训练集与测试集的划分是随机的，若一个人在训练集中只有 2 张照片则会出现这种现象。



(a)



(b)



(c)

图 5 人脸识别结果

3.2 基于 PCA 的图像压缩

为了更好的展示在三个色道的压缩效果和总体压缩效果，后续的每张图我们都展示 RGB，R，G，B 四幅图像。原图的尺寸为 243×437，下图为原图的展示。

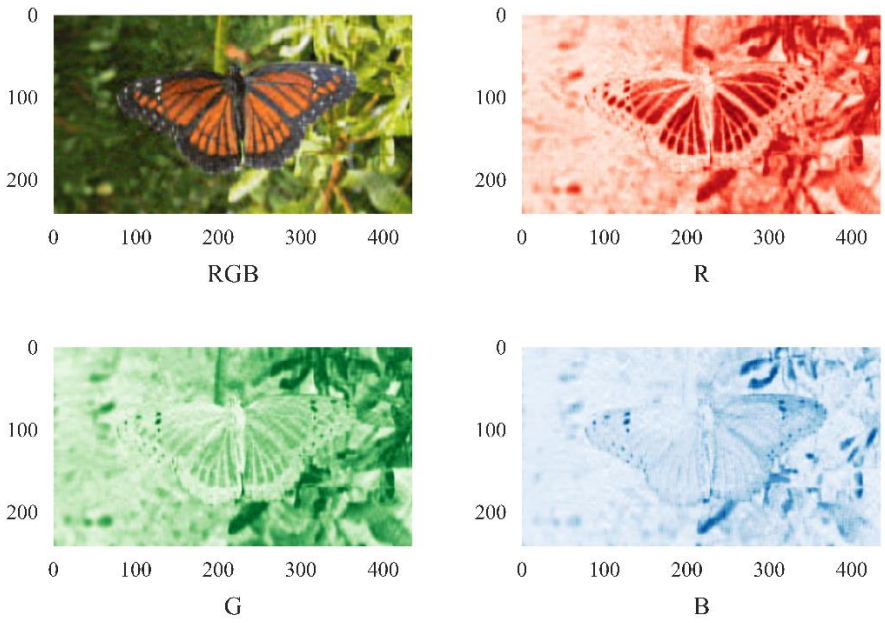
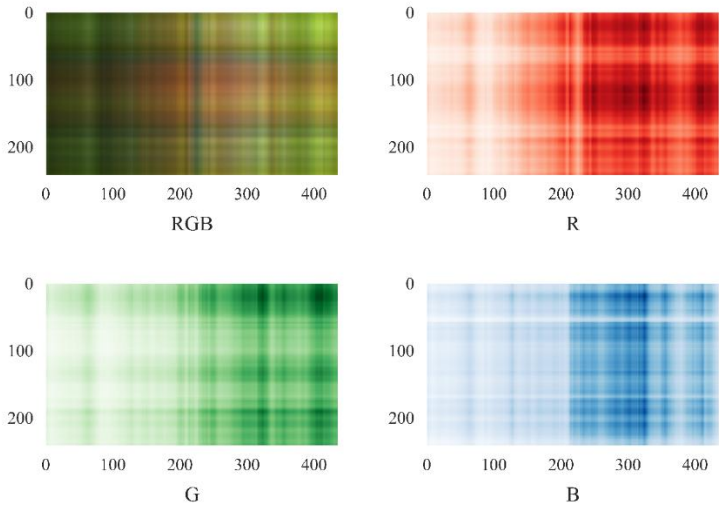
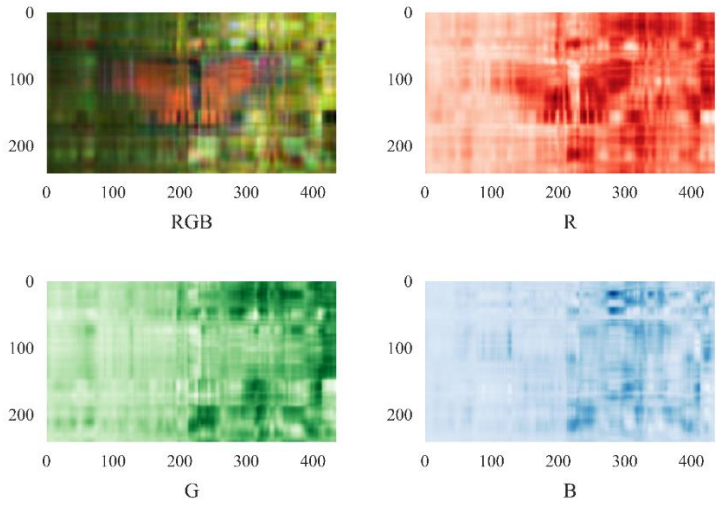


图 6 butterfly 原图

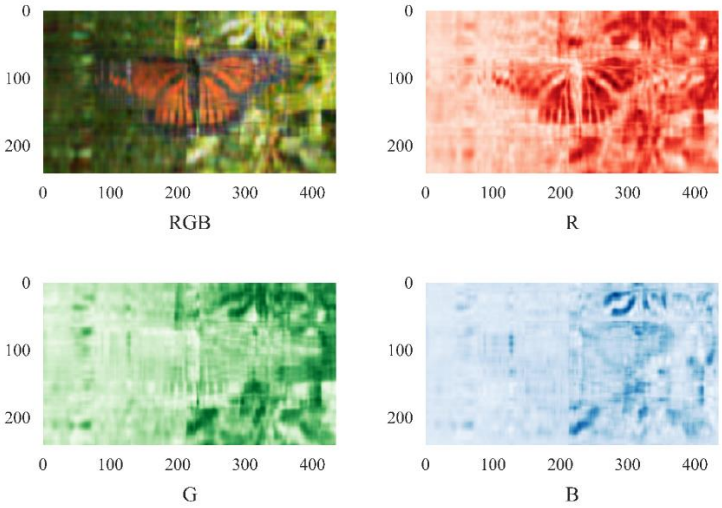
如果把每一行像素看作一个样本，则图片这个数据集有 237 个样本，437 个特征，作 PCA 得到 437 个主成分。选取不同数量的主成分对图像进行重构，选取主成分数量为 1，5，10，30 展示，效果如图 7。可见，在 k=5 时已经能大致看出蝴蝶外形，k=10 时能看到更多细节，k=30 时已经与原图无异。30 相比 437 是一个很小的值，不到 7%的主成分已经达到很好的效果。



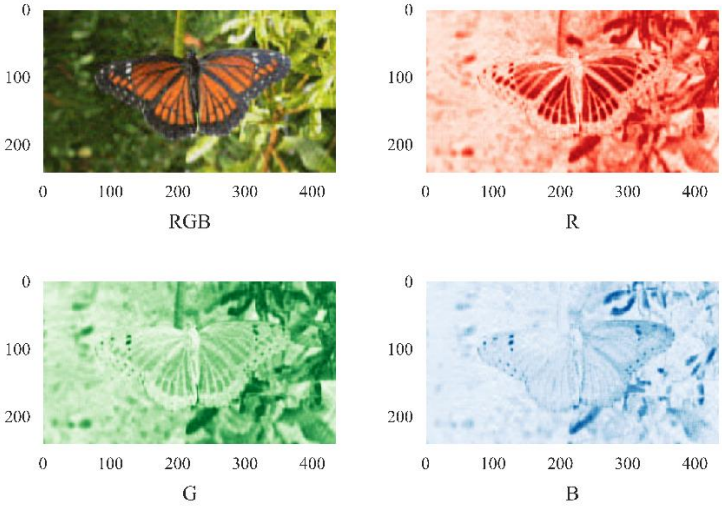
(a) k=1



(b) k=5



(c) k=10



(d) k=30

图 7 butterfly 图片的重构

展示前 49 个主成分如图 8。

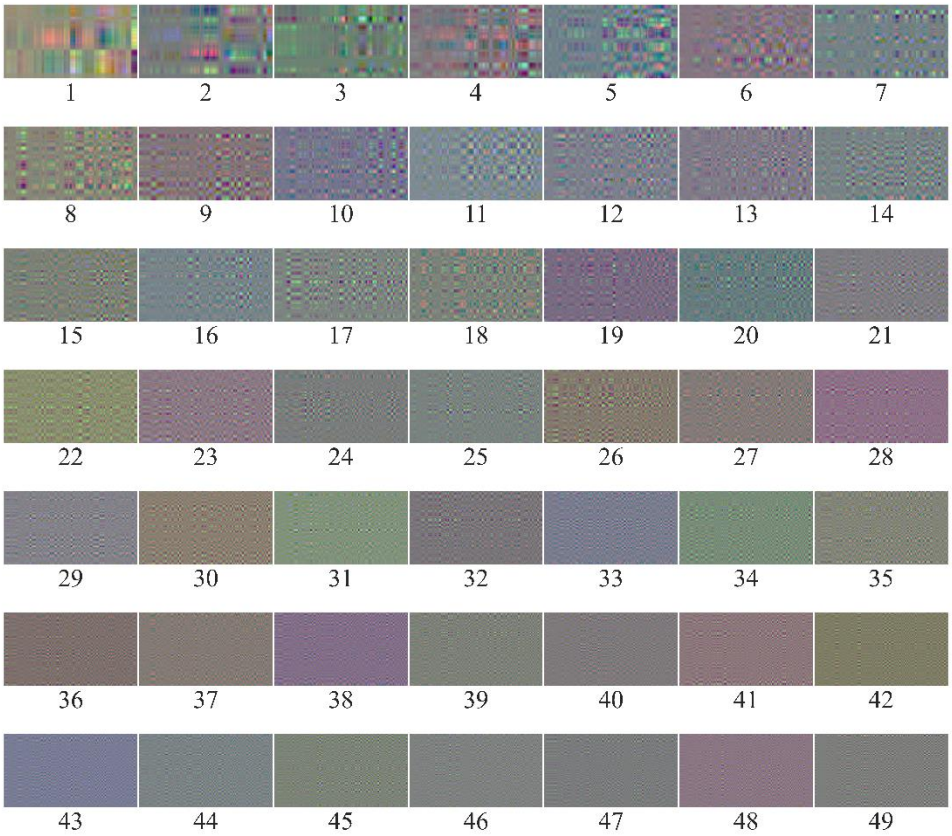


图 8 主成分

计算绝对误差与相对误差如图 9 所示，可以观察到误差随选取主成分的数量增加而迅速下降。

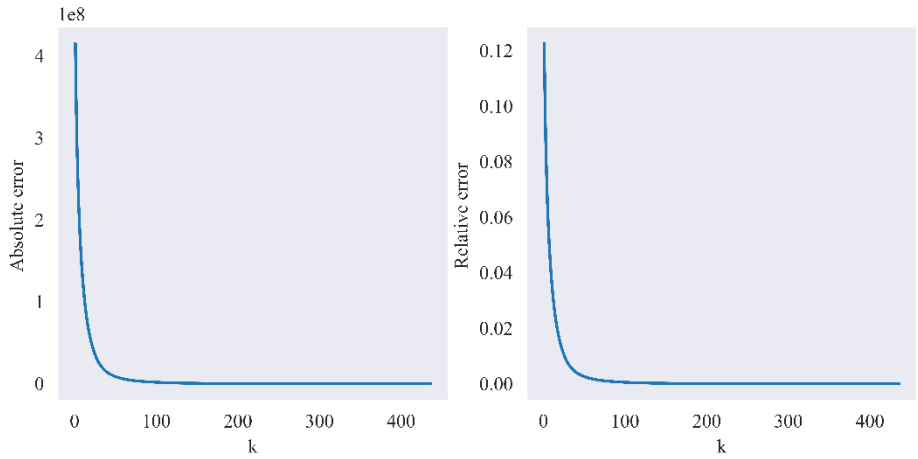
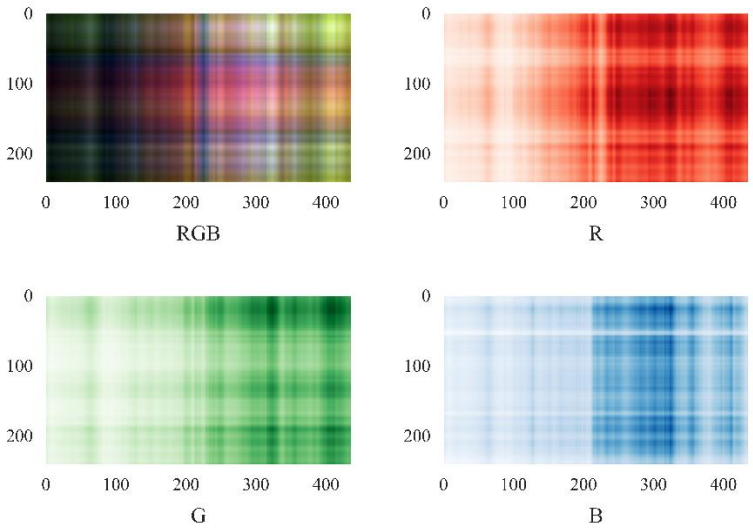


图 9 误差随选取主成分数量的变化

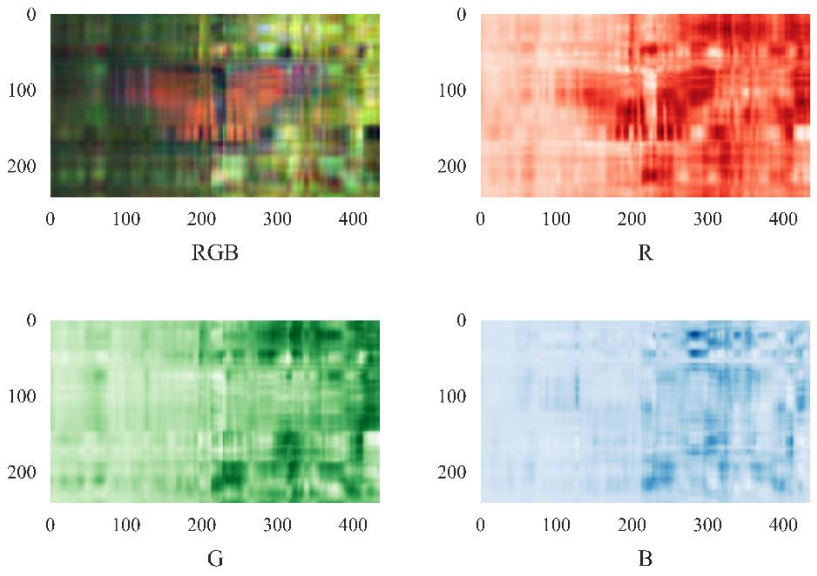
笔者还考察了将每一列像素当成一个样本时结果与上述结果有何差异，结论是无明显差异，此处不再展示。在本例中二者都符合物理含义，而据此可以推测，若将 3.1 的样本与特征互换，仍能得到很好的效果(尽管不再符合物理含义)。

3.3 基于 SVD 的图像压缩

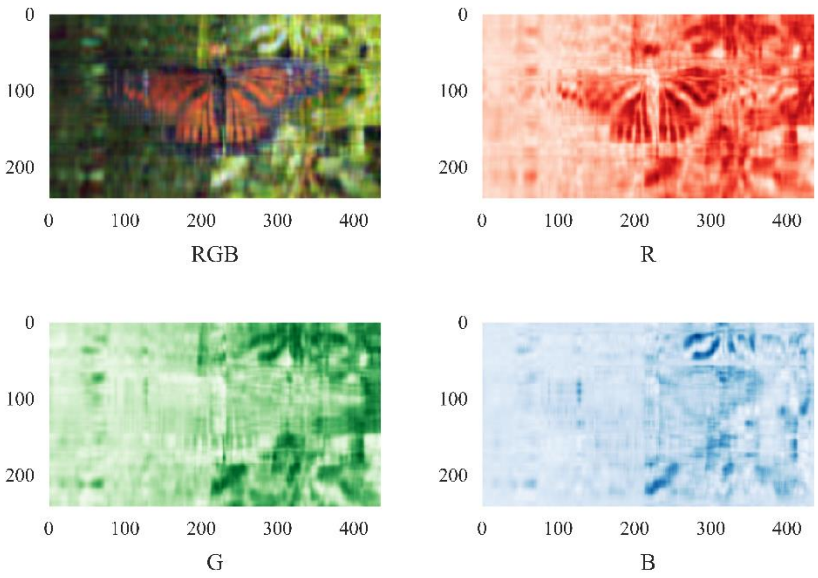
实验方法与 3.2 类似。考察左 k 个奇异向量、k 个奇异值、k 个右奇异向量重构图片的误差($k \leqslant 237$)。1，5，10，30 展示，效果如图 10。可见，与 3.2 类似，在 k=5 时已经能大致看出蝴蝶外形，k=10 时能看到更多细节，k=30 时已经与原图无异。30 相比 237 是一个很小的值，10%左右的向量已经达到很好的重构效果。



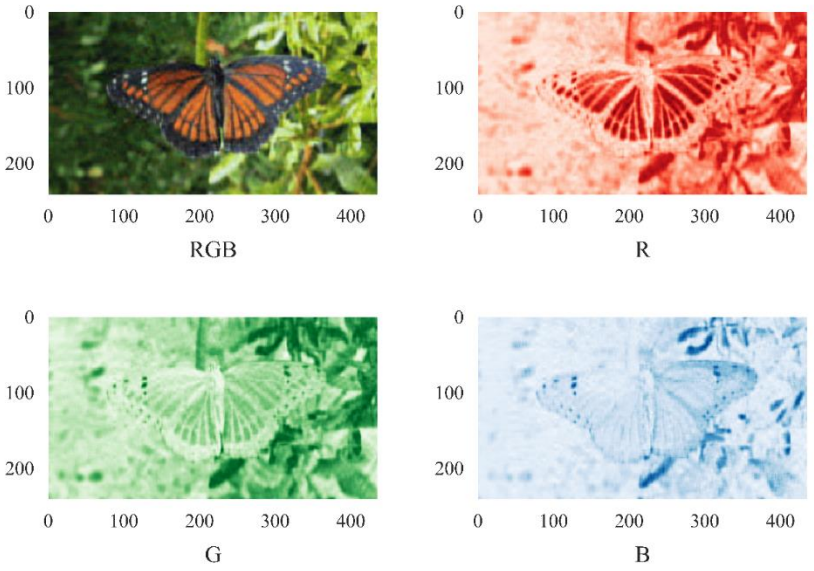
(a) k=1



(b) k=5



(c) k=10



(d) k=30

图 10 butterfly 图片的重构

展示前 49 个重构的图层如图 11。

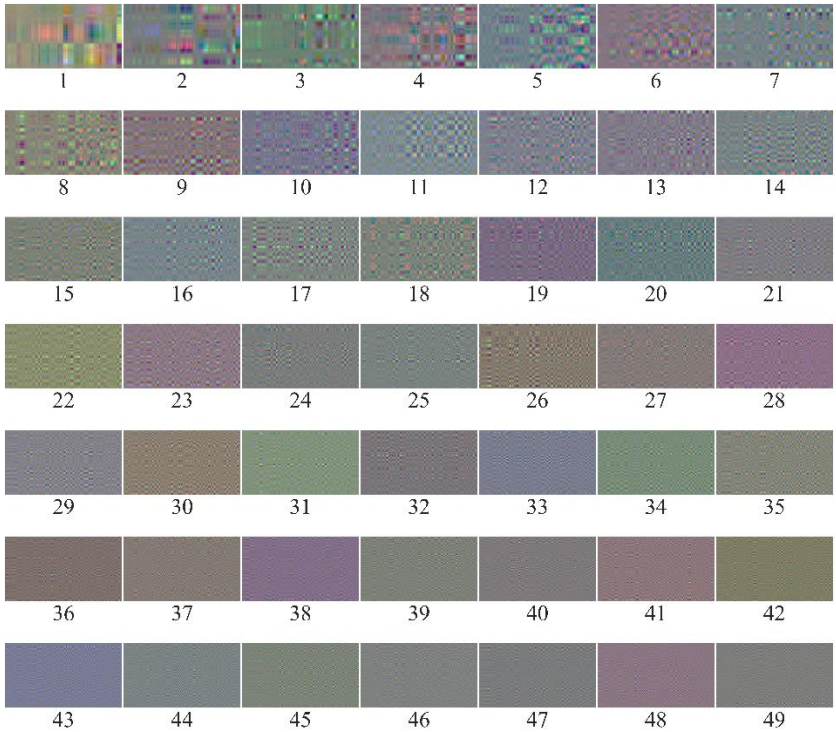


图 10 重构的图层

展示误差随图层数变化的趋势如图 11。

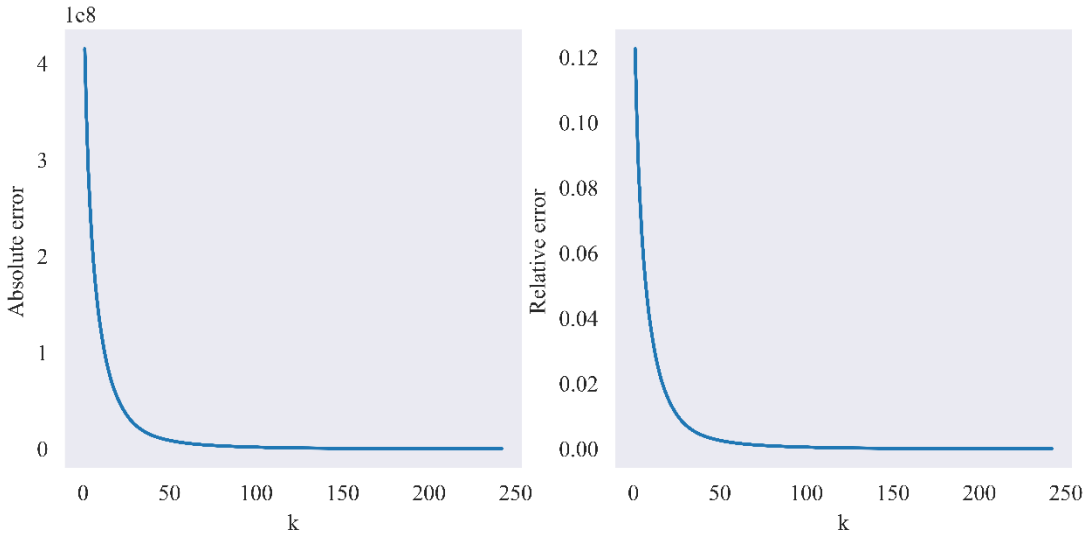


图 11 误差随选取主成分数量的变化

四 总结体会

4.1 总结

在本次实验中，我下载了 ORL 数据集，计算平均脸，通过 PCA 找到特征脸，并对数据进行降维、重构，也通过降维后的数据进行人脸识别，都得到很好的效果。同时，对 butterfly 图片分别使用 PCA、SVD 做降维、重构，都得到了比较好的效果，结果符合预期。

4.2 体会

虽然以前做过很多机器学习的实验，但本次实验是我第一次尝试不借助机器学习库而直接写代码。这个过程中遇到了诸多问题。

我尝试从矩阵运算开始写，但一方面计算速度不如 numpy 库，另一方面完成很多接口的彼此调用需要大量的时间，所以最终还是采用了 numpy 库。在探索 PCA 的过程中，遇到了很多问题，尝试了各种解决方案，从复数特征值到 sympy、matlab engine，从速度慢到阅

读 scipy 源码、使用 svd 来得到特征值、特征向量。对图像的数据结构、PIL 库不熟悉，导致 debug 了很多时间。

当写完 PCA 的整个搭建、测试框架，并成功跑通、复现 PPT 的内容后，还是感到很欣慰。之后再完成 SVD 的实验也就很轻松了，改动的代码其实不太多。

实验过程中学习到了很多，也有一些小的成就感。相信有了第一次实验的经验，后续会更熟练。