ML Homework 7

Bo-Cheng Huang, 309652022

1 Code with detailed explanations

1.1 Kernel Eigenfaces

For easier implementation, I resize all images from $(231, 195)$ to $(80, 68)$. The codes below are aimed to resize images.

```
train_list=glob.glob(os.path.join('Training','*.pgm'))
test_list=glob.glob(os.path.join('Testing','*.pgm'))
for i in range(len(train_list)):
    img=Image.open(train_list[i])
    img.thumbnail((80,80))
    img.save(train_list[i])
for i in range(len(test_list)):
    img=Image.open(test_list[i])
    img.thumbnail((80,80))
    img.save(test_list[i])
```

Listing 1: Resize images

Using X and Y to represent the ndarray form of training data and testing data, respectively. (training_data and testing_data are lists of images with size $80 \times 68 = 5440$ stored by grayscale)

```
h=80
w=68
n=len(train_data)
X=np.zeros((n,h*w))
for i in range(n):
    X[i]=train_data[i]
m=len(test_data)
Y=np.zeros((m,h*w))
for i in range(m):
    Y[i]=test_data[i]
```

Listing 2: Set training data X and testing data Y

In Part 1, for PCA method, I compute the eigenvector of covariance matrix of X, then use the first 133 eigenvectors with larger eigenvalue as projection matrix W in Listing 3. The first 25 eigenfaces are columns of W in Listing 4 and randomly pick 10 images to show their reconstruction by PCA. In Listing 6, Z is the reduction of training data after projection and YY is the reconstruction of training data.

```
X=X.T
X_m=np.mean(X,axis=1).reshape(-1,1)
X_c=np.subtract(X,X_m)
Cov=1/135*X_c@X_c.T
eigenvalue,eigenvector=np.linalg.eigh(Cov)
eigenvector/=np.linalg.norm(eigenvector,axis=0)
sort_index=np.argsort(-eigenvalue)
W=eigenvector[:,sort_index[:133]]
```

Listing 3: PCA

```
plt.figure(figsize=(8,10))
for i in range(25):
        plt.subplot(5,5,i+1)
        plt.axis('off')
        plt.imshow(W[:,i].reshape(h,w),cmap='gray')
```

Listing 4: Eigenfaces

```
rd_list=[]
for i in range(10):
    while 1:
        ind=rd.randint(0,n)
        if rd_list.count(ind)==0:
            rd_list.append(ind)
            break
```

Listing 5: Make a list to reconstruct images

```
Z=W.T@X_c
YY=W@Z+X_m
plt.imshow(YY[:,rd_list[i]].reshape(h,w),cmap='gray')
```

Listing 6: Reconstruction of images by PCA (example for ith in rd_list)

For LDA method, I divided train images into 30 classes where new_label_num contaning the number of items in each class (Line 3-8) so that I can get at least 25 fisherfaces. Before doing LDA, I compute within-class scatter S_w and between-class scatter S_b where

$$S_w = \sum_{j=1}^{k} S_j = \sum_{j=1}^{k} \sum_{i \in C_j} (x_i - \mathbf{m}_j)(x_i - \mathbf{m}_j)^T,$$

$$S_b = \sum_{j=1}^{k} S_{b_j} = \sum_{j=1}^{k} (\mathbf{m}_j - \mathbf{m})(\mathbf{m}_j - \mathbf{m})^T$$

where $\mathbf{m}_j = \frac{1}{n_j} \sum_{i \in C_j} x_i$ for all $j = 1, \ldots, k$ (Line 9-11) and $\mathbf{m} = \frac{1}{n} \sum x$ (Line 16).

```
1  list_X=[]
2  list_face_m=[]
3  for i in range(30):
4      X_temp=np.zeros((int(new_label_num[i]),h*w))
5      m=int(np.sum(new_label_num[:i]))
6      for j in range(m,m+int(new_label_num[i])):
7          X_temp[j-m]=train_data[j]
8      list_X.append(X_temp)
9  for i in range(30):
10     A=np.mean(list_X[i],axis=0)
11     list_face_m.append(A)
12 Sw=np.zeros((h*w,h*w))
13 for i in range(30):
14     X_c=np.subtract(list_X[i],list_face_m[i])
15     Sw+=X_c.T@X_c
16 m=1/135*np.sum(train_data,axis=0)
17 Sb=np.zeros((h*w,h*w))
18 for i in range(30):
19     X_c=np.subtract(list_face_m[i],m).reshape(-1,1)
20     Sb+=new_label_num[i]*X_c@X_c.T
```

Listing 7: Compute S_w and S_b

To handle with the sigularity of S_w, I use the algorithm in paper to map S_w into the range (subspace) of S_b. [1] First, since the rank of S_b is at most 29, I took the first 29 eigenvectors with largest eigenvalue. (others are in null space of S_b) Then use them to make matrix Y such that $Y^T S_b Y = D_b$, where D_b contains those eigenvalues. Let $Z = Y D_b^{-\frac{1}{2}}$, then $Z^T S_b Z$ is identity. Consider $Z^T S_w Z$ and its eigenvectors, I took those with nontrivial eigenvalue of $Z^T S_w Z$ to make a matrix U. So $U^T Z^T$ is the desired result.

[1] Yu, H.; Yang, J. (2001). "A direct LDA algorithm for high-dimensional data — with application to face recognition", Pattern Recognition, 34 (10), 2067–2069

```
eigenvalue,eigenvector=np.linalg.eigh(Sb) # rank 29
sort_index=np.argsort(-eigenvalue)
Y=eigenvector[:,sort_index[:29]]
Db=np.diag(eigenvalue[sort_index[:29]])
Db_inverse_square_root=np.diag(1/np.diag(np.sqrt(Db)))
Z=Y@Db_inverse_square_root
eigenvalue,eigenvector=np.linalg.eigh(Z.T@Sw@Z)
sort_index=np.argsort(-eigenvalue)
U=eigenvector[:,sort_index]
A=U.T@Z.T
Dw=np.diag(np.diag(A@Sw@A.T))
```

Listing 8: LDA

The first 25 fisherfaces are the rows of $A = U^T Z^T$ in Listing 9 and randomly pick 10 images to show their reconstruction by LDA, where $D_w^{-\frac{1}{2}} Ax$ is the reduction of training data and $A^T D_w^{-1} Ax$ is the reconstruction of training data.

```
plt.figure(figsize=(8,10))
for i in range(25):
        plt.subplot(5,5,i+1)
        plt.axis('off')
        plt.imshow(A[i,:].reshape(h,w),cmap='gray')
P=A.T@np.linalg.inv(Dw)@A@train_data[rd_list[i]].reshape(-1,1)
plt.imshow(P.reshape(h,w),cmap='gray')
```

Listing 9: Fisherfaces and reconstruction of images by LDA (example for ith in rd_list)

In Part 2, I define a function to compute the accurancy of testing data in PCA and LDA using k nearest neighbor. (Set $k = 3$)

```
# PCA
Z1=W.T@(X-X_m)
Z2=W.T@(Y-X_m)
# LDA
Z1=np.diag(1/np.diag(np.sqrt(Dw)))@A@X
Z2=np.diag(1/np.diag(np.sqrt(Dw)))@A@Y
```

Listing 10: Reduction of X and Y

```
label=[]
for i in range(135):
    for j in range(15):
        if face_exp[j] in train_list[i]:
            label.append(j)
            break
label2=[]
for i in range(30):
    for j in range(15):
        if face_exp[j] in test_list[i]:
            label2.append(j)
            break
```

Listing 11: Label of training data and testing data

```
def accurancy(Z1,Z2):
    '''
    Z1: Training data in new space
    Z2: Testing data in new space
    Using k-NN (k=3) to determine the label of Testing data
    '''
    guess=[]
    for i in range(30):
        dist=Z1-Z2[:,i].reshape(-1,1)
        dist=np.sum(np.power(dist,2),axis=0)
        index=np.argsort(dist)
        list1=[]
        for j in range(3):
            list1.append(label[index[j]])
        labelcount=np.zeros(15)
        for j in range(15):
            labelcount[j]=list1.count(j)
        if np.max(labelcount)>1:
            guess.append(np.argmax(labelcount))
        else:
            guess.append(list1[0])
    T=0
    for i in range(30):
        if label2[i]==guess[i]:
            T+=1
    print(T/30)
```

Listing 12: Accurancy of model

In Part 3, I use some Kernel functions to implement kernel PCA and kernel LDA.

- linear: $K(x_i, x_j) = x_i^T x_j$.
- polynomial: $K(x_i, x_j) = (\gamma x_i^T x_j)^d,\ \gamma > 0$.
- radial basis function: $K(x_i, x_j) = \exp(-\gamma||x_i - x_j||^2),\ \gamma > 0$.

```
def kernel(X1,X2,gamma,d,mode):
    '''
    Impletation of kernel
    X1: (n,h*w)-array
    X2: (m,h*w)-array
    gamma: parameters
    return a (n,m)-array
    '''
    if mode==0:
        return X1@X2.T
    elif mode==1:
        return np.power(gamma*X1@X2.T,d)
    else:
        XY=np.zeros((np.size(X1,0)+np.size(X2,0),h*w))
        XY[:np.size(X1,0),:]=X1
        XY[np.size(X1,0):,:]=X2
        K=dis.squareform(np.exp(-gamma*dis.pdist(XY,'sqeuclidean')))
        return K[:np.size(X1,0),np.size(X1,0):]
```

Listing 13: Kernel function

For kernel PCA in Listing 14, first I defined the kernel K of training data such that $K_{ij} = k(x_i, x_j)$. Then compute the eigenvectors of the centralized kernel matrix K_c and use the first 134 eigenvectors to obtain a projection matrix W related to K.

For kernel LDA in Listing 15, different from kernel PCA, I need to compute kernel between all 30 classes of training data. Let the rth sample of class t and the sth sample of class u be x_{tr} and x_{us}, respectinely. Then $(k_{rs})_{tu} = k(x_{tr}, x_{us})$. The kernel matrix K is composed of those small kernel K_{tu} for $1 \leq t, u \leq 30$. Let $Z = (Z_t)$ where Z_t is a $l_t \times l_t$ matrix with terms all equal to $\frac{1}{l_t}$ for $1 \leq t \leq 30$, i.e., Z is a 135×135 block diagonal matrix. Then I obtain a projection matrix A by solving the first 50 eigenvectors of $K_c^{-1} Z K_c$.[2]

[2]M.-H. Yang, "Kernel Eigenfaces vs. Kernel Fisherfaces: Face recognition using kernel methods," Proceedings of Fifth IEEE International Conference on Automatic Face Gesture Recognition, 2002, pp. 215-220

```
id_M=1/135*np.identity(135)
K=kernel(X,X,1e-8,0,2)
K_c=K-id_M@K-K@id_M+id_M@K@id_M
eigenvalue,eigenvector=np.linalg.eigh(K_c)
eigenvector/=np.linalg.norm(eigenvector,axis=0)
sort_index=np.argsort(-eigenvalue)
W=eigenvector[:,sort_index[:134]]
id_M_new=1/135*np.ones((135,30))
K_new=kernel(X,Y,1e-8,0,2)
K_new_c=K_new-K@id_M_new-id_M@K_new+id_M@K@id_M_new
Z1=W.T@K #K_c
Z2=W.T@K_new #K_new_c
```

Listing 14: kernel PCA (with RBF kernel)

```
K=np.zeros((135,135))
for i in range(30):
    for j in range(30):
        xi=int(np.sum(new_label_num[:i]))
        yi=int(new_label_num[i])
        xj=int(np.sum(new_label_num[:j]))
        yj=int(new_label_num[j])
        K[xi:xi+yi,xj:xj+yj]+=kernel(list_X[i],list_X[j],1,0,0)
K_c=K-id_M@K-K@id_M+id_M@K@id_M
Z=np.zeros((135,135))
for i in range(30):
    m=int(np.sum(new_label_num[:i]))
    for j in range(m,m+int(new_label_num[i])):
        for k in range(m,m+int(new_label_num[i])):
            Z[j][k]=1/new_label_num[i]
eigenvalue,eigenvector=np.linalg.eigh(np.linalg.inv(K_c)@Z@K_c)
sort_index=np.argsort(-eigenvalue)
A=eigenvector[:,sort_index[:50]]
id_M_new=1/135*np.ones((30,135))
K_new=kernel(X,Y,1,0,0)
K_new=K_new.T
K_new_c=K_new-id_M_new@K-K_new@id_M+id_M_new@K@id_M
Z1=A.T@K #K_c
Z2=A.T@K_new #K_new_c
```

Listing 15: kernel LDA (with linear kernel)

1.2 t-SNE

In Part 1, for symmetric SNE, it uses a simpler cost function based on a single KL divergence between a joint probability distribution.

$$p_{ij} = \frac{\exp(-\|x_i - x_j\|^2/(2\sigma^2))}{\sum_{k\neq l}\exp(-\|x_l - x_k\|^2/(2\sigma^2))},$$
$$q_{ij} = \frac{\exp(-\|y_i - y_j\|^2)}{\sum_{k\neq l}\exp(-\|y_l - y_k\|^2)}.$$

For t-SNE, p_{ij} remains and it changes Gaussian distribution to Student t-distribution in low dimension.

$$q_{ij} = \frac{(1 + \|y_i - y_j\|^2)^{-1}}{\sum_{k\neq l}(1 + \|y_l - y_k\|^2)^{-1}}.$$

The main difference between symmetric SNE and t-SNE is the distribution of low dimension data, so I modify the codes related to $Q = (q_{ij})$ and gradient dY ($\frac{\partial C}{\partial y_i} = \sum_j (p_{ij} - q_{ij})(y_i - y_j)(1 + \|y_i - y_j\|^2)^{-1}$) in Listing 16 and ($\frac{\partial C}{\partial y_i} = \sum_j (p_{ij} - q_{ij})(y_i - y_j)$) in Listing 17.

```
# Compute pairwise affinities
sum_Y = np.sum(np.square(Y), 1)
num = -2. * np.dot(Y, Y.T)
num = 1. / (1. + np.add(np.add(num, sum_Y).T, sum_Y))
num[range(n), range(n)] = 0.
Q = num / np.sum(num)
Q = np.maximum(Q, 1e-12)

# Compute gradient
PQ = P - Q
for i in range(n):
    dY[i, :] = np.sum(np.tile(PQ[:, i] * num[:, i], (no_dims, 1)).T * (Y[i, :] - Y), 0)
```

Listing 16: Difference in t-SNE

In Part 2, I add some codes in this part to save a figure of data every 10 iterations in Listing 18. Then using function to generate GIF in Listing 19. (same as HW6)

In Part 3, I show the discrete distribution of pairwise similarities in both high-dimensional space (P) and low-dimensional space (Q) in Listing 20.

In Part 4, I only need to change the parameter 'perplexity' in tsne and symsne.

```
# Compute pairwise affinities
sum_Y = np.sum(np.square(Y), 1)
num = -2. * np.dot(Y, Y.T)
num = np.exp(-np.add(np.add(num,sum_Y).T,sum_Y))
num[range(n), range(n)] = 0.
Q = num / np.sum(num)
Q = np.maximum(Q, 1e-12)

# Compute gradient
PQ = P - Q
for i in range(n):
    dY[i, :] = np.sum(np.tile(PQ[:, i],(no_dims, 1)).T*(Y[i, :] - Y),0)
```

Listing 17: Difference in symmetric SNE

```
# Compute current value of cost function
        if (iter + 1) % 10 == 0:
            C = np.sum(P * np.log(P / Q))
            print("Iteration %d: error is %f" % (iter + 1, C))
            plt.scatter(Y[:, 0], Y[:, 1], 20, labels)
            plt.savefig('new_'+str((iter + 1) // 10)+'.png')
            plt.clf()
```

Listing 18: Save the scatter 10 iterations

```
imagelist=[]
for i in range(40):
    imagelist.append('new_'+str(i)+'.png')
with imageio.get_writer('newgif.gif', mode='I') as writer:
    for filename in imagelist:
        image = imageio.imread(filename)
        writer.append_data(image)
```

Listing 19: Making GIF

```
def plot_similarity(P,Q):
    plt.subplot(2,1,1)
    plt.hist(P.flatten(),bins=50,color='blue',density=True,histtype='step',log=True)
    plt.subplot(2,1,2)
    plt.hist(Q.flatten(),bins=50,color='green',density=True,histtype='step',log=True)
    plt.show()
```

Listing 20: Visualization of the distributions

2 Experiments settings and results

2.1 Kernel Eigenfaces

In Part 1, there are the first 25 eigenfaces and 25 fisherfaces in Figure 1. Some eigenfaces are related to someone's face but fisherfaces are just expressed contour of faces. In Figure 2, from the reconstruction of images by PCA, it is similar to the original image. But by LDA it only keeps facial features, not the whole face.

Figure 1: 25 eigenfaces (left) and 25 fisherfaces (right)

Figure 2: Reconstruct 10 images by PCA (left) and LDA (right)

In Part 2, the accurancy of PCA is 86.7% and the accurancy of LDA is 100% based on the algorithm I chose.

In Part 3, the accurancy of kernel PCA with uncentralized linear kernel is 73.3% but the accurancy of kernel PCA with centralized linear kernel is just 0.67%. By kernel PCA with uncentralized polynomial kernel the best performance is 76.7% but the best performance of kernel PCA with centralized polynomial kernel is 26.7% with $d = 5$, the influence of γ is weak. (I tried γ from 10^{-10} to 10^{-2} and $d = 2, 3, 4, 5$) By kernel PCA with RBF kerne, I tried γ from 10^{-10} to 10^{-5}, the best performance of uncentralized kernel is 83.4% for 10^{-9} and the best performance of centralized kernel is 76.7% for 10^{-6}.

kernel PCA	linear	polynomial	RBF
uncentralized	73.3	76.7	83.4
centralized	0.67	26.7	76.7

Table 1: Best performance of different kernels in kernel PCA

The accurancy of kernel LDA with uncentralized linear kernel is 76.7% (greater than PCA) but the accurancy of kernel LDA with centralized linear kernel is 0.67%. By kernel LDA with uncentralized polynomial kernel the best performance is 76.7% but the best performance of kernel LDA with centralized polynomial kernel is 30% with $d = 4$ (greater than PCA), the influence of γ is weak.(I tried γ from 10^{-10} to 10^{-2} and $d = 2, 3, 4$) By kernel LDA with RBF kerne, I tried γ from 10^{-10} to 10^{-5}, the best performance of uncentralized kernel is 83.4% for 10^{-10} and the best performance of centralized kernel is 66.7% for 10^{-7}.

kernel LDA	linear	polynomial	RBF
uncentralized	76.7	76.7	83.4
centralized	0.67	30	66.7

Table 2: Best performance of different kernels in kernel LDA

From the experiment of Part 2 to Part 3, original PCA and LDA have better accurancy than kernel PCA and kernel LDA's. I observe that if the kernel is uncentralized, then the influence of parameters in kernel is weaker than centralized kernel, but the centralized kernel has bad performance since the centralized kernel of training data and testing data K_new_c may contain negative number such that the choice of kernel function and related parameters is importanat. It needs to be careful to select parameters for centralized kernel.

2.2 t-SNE

In Part 1, for the symmetric SNE method, it is hard to recongnize of all 10 clusters since the distance between each pair of clusters is too close in low dimension, so the crowding problem occurs. To solve this problem, using Student t-distribution instead of Gaussian distribution in low dimension is a good choice. Hence the 10 clusters are far away each other in t-SNE method.

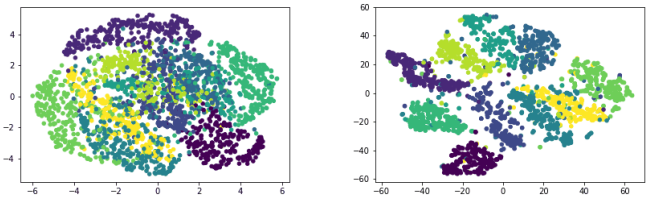

Figure 3: The result of symmetric SNE (left) and t-SNE (right)

In Part 2, there are GIFs of the optimize procedure in symmetric SNE and t-SNE. For the first 100 iterations, it wants to separate all clusters using lying P. After 100 iterations it changes to the true P. (click it will move)

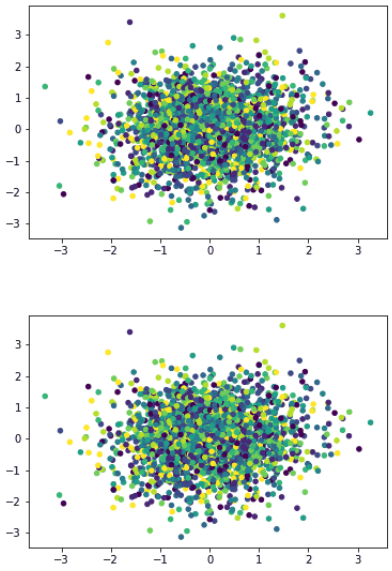

Figure 4: The GIFs of symmetric SNE (top) and t-SNE (bottom)

In Part 3, for the distribution of pairwise similarities in low-dimensional space, the value of t-SNE is greater than symmetric SNE's ($10^{-5} > 10^{-6}$) based on different distribution. Hence using Student t-distribution would relax more pairwise similarities.

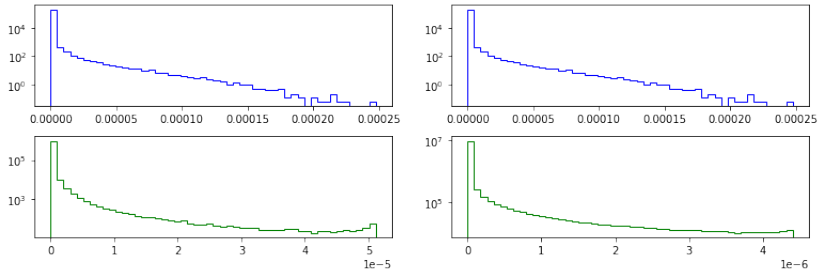

Figure 5: The distribution of pairwise similarities in high-dimensional (blue) and low-dimensional (green), based on t-SNE (left) and symmetric SNE (right)

In Part 4, I tried the cases of perplexity for $5, 20, 50$ and the results are below. If I use large perplexity, then the distances between data points are smaller. For t-SNE, using smaller perplexity (5) would make the data to be loose, so some clusters are not gathering, while using lagerer perplexity (50) would make the data in same cluster to be tighter and there is one cluster cut by other data points in $x \in (-40, 0)$ and $y \in (-10, 20)$.

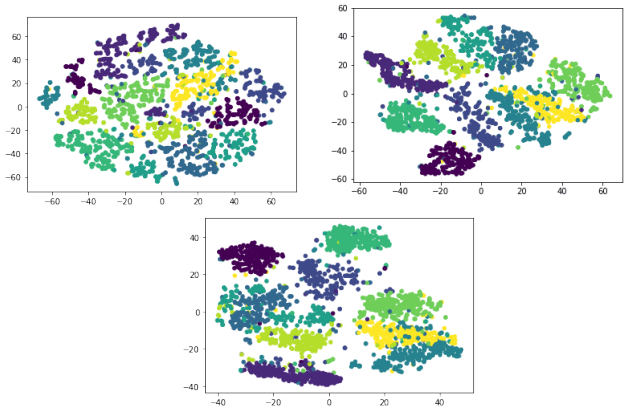

Figure 6: The result of t-SNE with perplexity $5, 20, 50$

For symmetric SNE, since the distribution of pairwise similarities is Gaussian distribution in low dimension space, it is not clear to observe the difference using different perplexity. But the distance between data points are also smaller by

large perplexity.

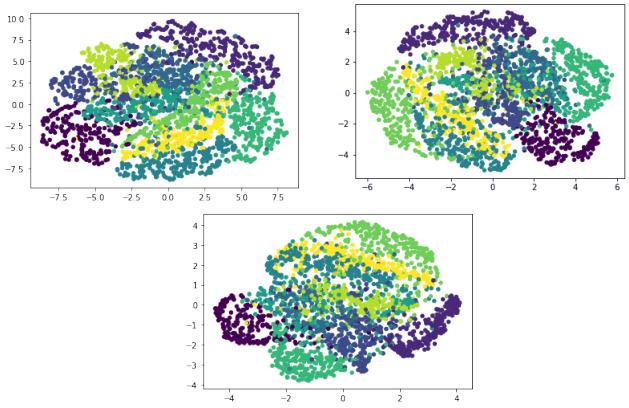

Figure 7: The result of symmetric SNE with perplexity $5, 20, 50$

3 Observations and discussion

For the first part of homework 7, since both PCA and LDA are based on matrix calculation, we cannot keep all the information of images, so there is a trade-off to select the size and scale of images. To do face recognition, using grayscale instead of RGB is enough to save the major imformation of face features.

In PCA, it needs to deal with the eigenvalue problem of large covariance matrix, then use those eigenvectors to do reduction. In LDA, it needs to compute 2 large matrices S_w and S_b. Since the number of images is typically smaller than the dimensionality of the images, both matrices are singular. By the algorithm I used in Part 2, it has good performance than using the pseodo-inverse of S_w.

In kernel method of both PCA and LDA, it deducts lots of computation since the size of kernel is based on the number of images, so it is faster and easier than using original PCA and LDA. But by kernel method, it reduct strictly so there are some imformation missing such that the centralized kernel may not perform well.

For the second part of homework 7, it is a good chance to know how to implement t-SNE. Before compute p_{ij}, it needs to find σ to compare with all pairwise distance. Then find the gradient of KL divergences $\mathrm{KL}(P||Q)$ and modify P and Q.