HW07

資工碩一 吳承翰 0856105

1. Eigenface and fisherface

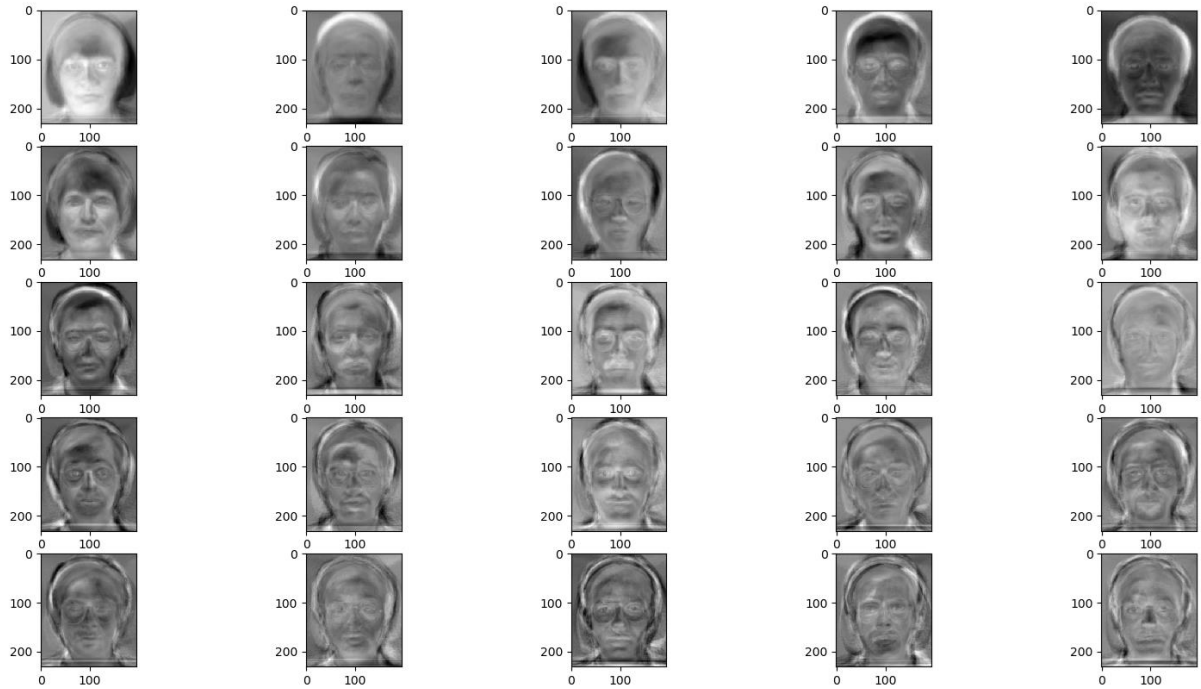
eigenface:
Because X@X.T will be a very large matrix，we find the eigenvalues & eigenvectors of X.T@X matrix，these corresponding eigenvectors(without eigenvalue smaller than 0) then multiply X again representing a partial eigenvectors of X@X.T.
Doing PCA to reduce dimension from 231*195 -> 134.

```
def pca(X,num_dim=None):
    X_mean = np.mean(X, axis=1).reshape(-1, 1)
    X_center = X - X_mean

    # PCA
    eigenvalues, eigenvectors = np.linalg.eig(X_center.T @ X_center)
    sort_index = np.argsort(-eigenvalues)
    if num_dim is None:
        for eigenvalue, i in zip(eigenvalues[sort_index], np.arange(len(eigenvalues))):
            if eigenvalue <= 0:
                sort_index = sort_index[:i]
                break
    else:
        sort_index=sort_index[:num_dim]

    eigenvalues=eigenvalues[sort_index]
    # from X.T@X eigenvector to X@X.T eigenvector
    eigenvectors=X_center@eigenvectors[:, sort_index]
    eigenvectors_norm=np.linalg.norm(eigenvectors,axis=0)
    eigenvectors=eigenvectors/eigenvectors_norm

    return eigenvalues,eigenvectors,X_mean
```

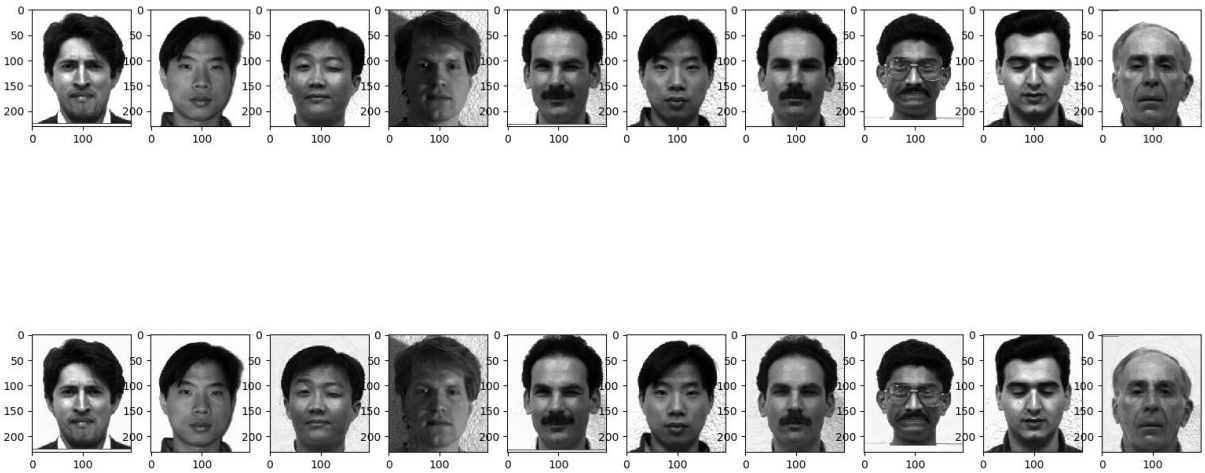
```
H,W=231,195
X,y=imread(filepath,H,W)

eigenvalues,eigenvectors,X_mean=pca(X)
# Transform matrix
U=eigenvectors.copy()
print('U shape: {}'.format(U.shape))

# show top 25 eigenface
show_eigenface(U,25,H,W)

# reduce dim (projection)
Z=U.T@(X-X_mean)

# recover
X_recover=U@Z+X_mean
show_reconstruction(X,X_recover,10,H,W)
```

U is the transformation matrix.
We can get eigenface from the columns of U.

When doing recovering，just project X into a lower dimension using U then project back to the original dimension using U.T.



Showing the eigenface from the columns of matrix U.



Showing the reconstructed result.

The first row is the original faces，the second row is the reconstructed faces.

When doing recovering，just project X into a lower dimension using U then project back to the original dimension using U.T，after that remember to add the mean of the X due to the preprocessing step(we center the X in beginning).

```python
# reduce dim (projection)
Z_test=U.T@(X_test-X_mean)

# k-nn
predicted_y=np.zeros(Z_test.shape[1])
for i in range(Z_test.shape[1]):
    distance=np.zeros(Z_train.shape[1])
    for j in range(Z_train.shape[1]):
        distance[j]=np.sum(np.square(Z_test[:,i]-Z_train[:,j]))
    sort_index=np.argsort(distance)
    nearest_neighbors=y_train[sort_index[:k]]
    unique, counts = np.unique(nearest_neighbors, return_counts=True)
    nearest_neighbors=[k for k,v in sorted(dict(zip(unique, counts)).items(), key=lambda item: -item[1])]
    predicted_y[i]=nearest_neighbors[0]

acc=np.count_nonzero((y_test-predicted_y)==0)/len(y_test)
```

Using the k-nn to predict the testing image label，in my case，I set k to be 3.

The accuracy is 83.33%.

```
C:\Anaconda3\python.exe C:/Users/
U shape: (45045, 134)
acc: 83.33%

Process finished with exit code 0
```

fisherface:

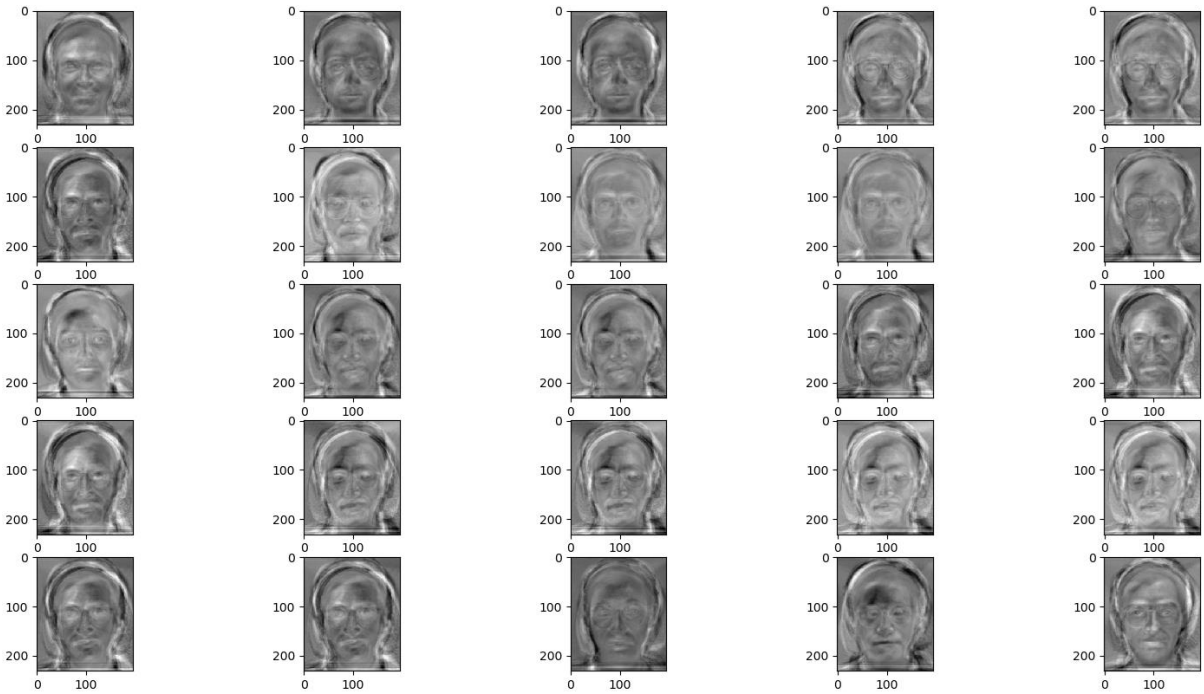
Because of the singularity of the within-class scatter matrix，I first do the PCA to reduce the dimension from 231*195 to a lower dimension，then do the LDA to get my result.

```
def lda(X,y,num_dim=None):
    N=X.shape[0]
    X_mean = np.mean(X, axis=1).reshape(-1, 1)

    classes_mean = np.zeros((N, 15))  # 15 classes
    for i in range(X.shape[1]):
        classes_mean[:, y[i]] += X[:, y[i]]
    classes_mean = classes_mean / 9

    # within-class scatter
    S_within = np.zeros((N, N))
    for i in range(X.shape[1]):
        d = X[:, y[i]].reshape(-1,1) - classes_mean[:, y[i]].reshape(-1,1)
        S_within += d @ d.T

    # between-class scatter
    S_between = np.zeros((N, N))
    for i in range(15):
        d = classes_mean[:, i].reshape(-1,1) - X_mean
        S_between += 9 * d @ d.T

    eigenvalues,eigenvectors=np.linalg.eig(np.linalg.inv(S_within)@S_between)
    sort_index=np.argsort(-eigenvalues)
    if num_dim is None:
        sort_index=sort_index[:-1]  # reduce 1 dim
    else:
        sort_index=sort_index[:num_dim]

    eigenvalues=np.asarray(eigenvalues[sort_index].real,dtype='float')
    eigenvectors=np.asarray(eigenvectors[:,sort_index].real,dtype='float')

    return eigenvalues,eigenvectors
```

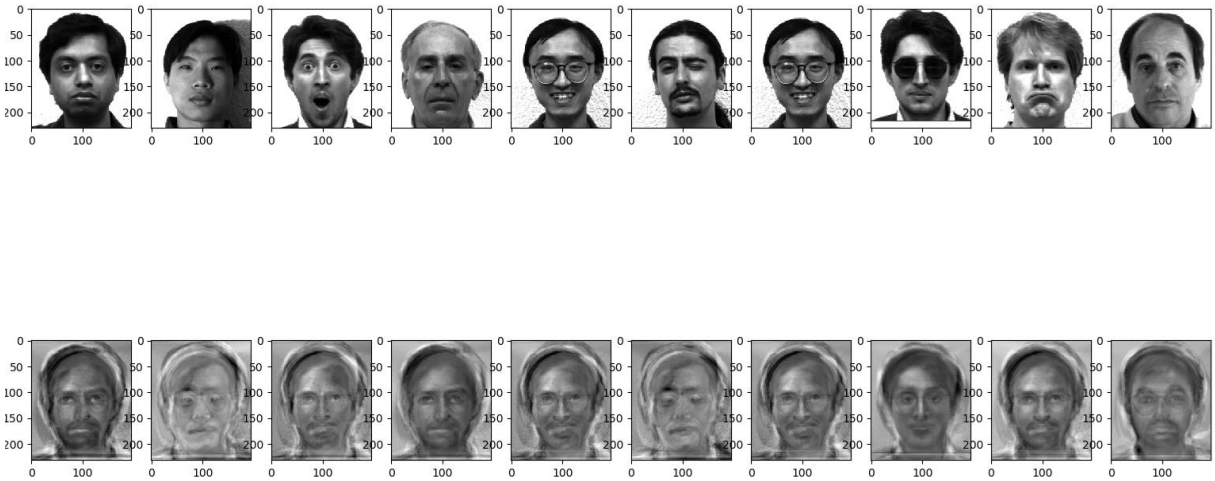
In my case，I first reduce the dimension to 31-dim by PCA，then reduce the dimension to 30-dim by LDA. The reason why I choose 31 as my objected dimension is by the final testing images’ accuracy，I test this value from 30~130.

The transformation matrix U is composed of eigenvector of PCA & eigenvector of LDA.

```
X,y=imread(filepath,H,W)

eigenvalues_pca,eigenvectors_pca,X_mean=pca(X,num_dim=31)
X_pca=eigenvectors_pca.T@(X-X_mean)
eigenvalues_lda,eigenvectors_lda=lda(X_pca,y)

# Transform matrix
U=eigenvectors_pca@eigenvectors_lda
print('U shape: {}'.format(U.shape))

# show top 25 eigenface
show_eigenface(U,25,H,W)

# reduce dim (projection)
Z=U.T@X

# recover
X_recover=U@Z+X_mean
show_reconstruction(X,X_recover,10,H,W)
```



Showing the fisherface from the columns of matrix U.



Showing the reconstructed result.

The first row is the original faces，the second row is the reconstructed faces.

When doing recovering，just project X into a lower dimension using U then project back to the original dimension using U.T，after that remember to add the mean of the X due to the preprocessing step(we center the X in beginning).

In my case，I set the k of k-nn to be 5.

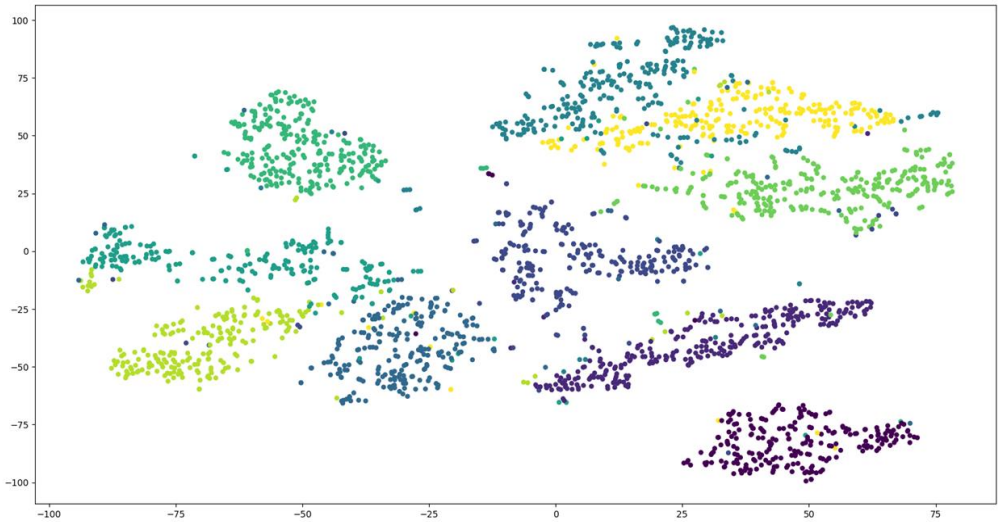
The final testing image accuracy is 70%.

```
C:\Anaconda3\python.exe C:/Users/
U shape: (45045, 30)
acc: 70.00%

Process finished with exit code 0
```

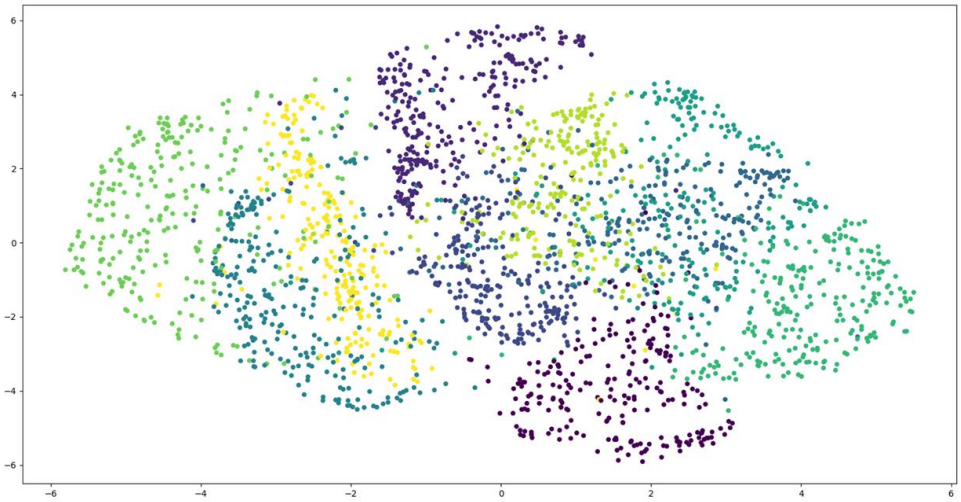
2.t-SNE

To change to symmetric SNE，just have to modify two lines of code.
The first one is “num=np.exp(-(num+.....))” from student-t distribution to pairwise similarity estimate by exponential of 2-norm，the second one is “dY[i,:]=np.dot(PQ……)” a different gradient descent from t-SNE.

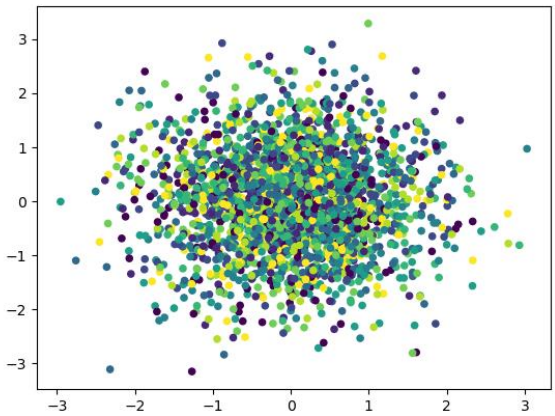
```
for iter in range(max_iter):

    # Compute pairwise affinities
    sum_Y = np.sum(np.square(Y), 1)
    num = -2. * np.dot(Y, Y.T)
    num=np.exp(-(num+sum_Y+sum_Y.reshape(-1,1)))
    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.dot(PQ[i,:],Y[i,:]-Y)

    # Perform the update
```

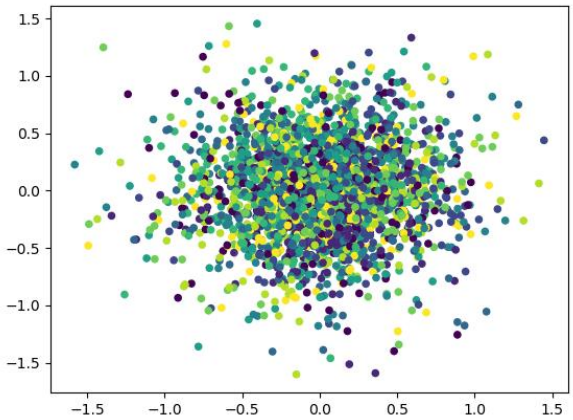


The t-SNE scatter plot after 500 iterations.



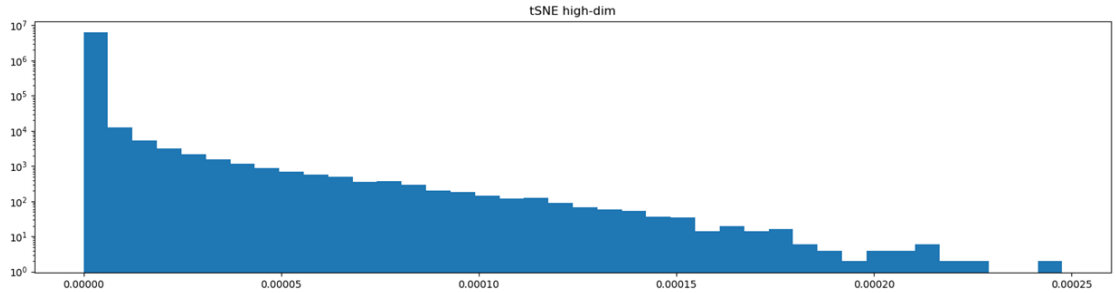
The symmetric SNE scatter plot after 500 iterations.

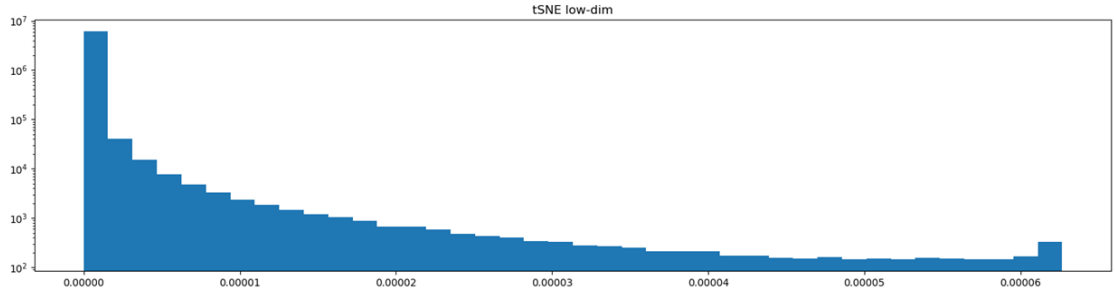


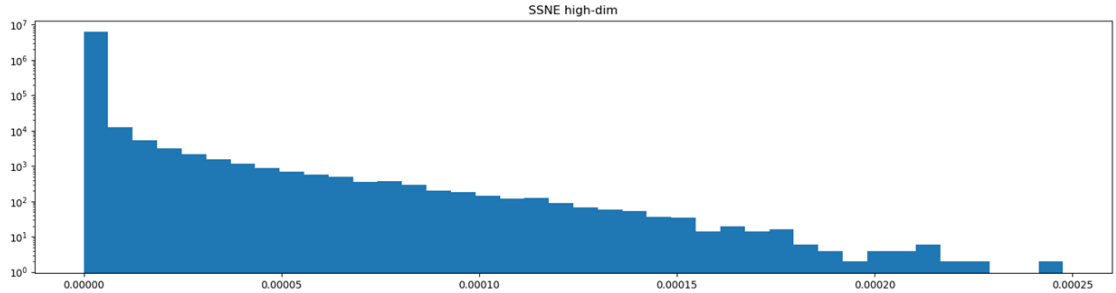


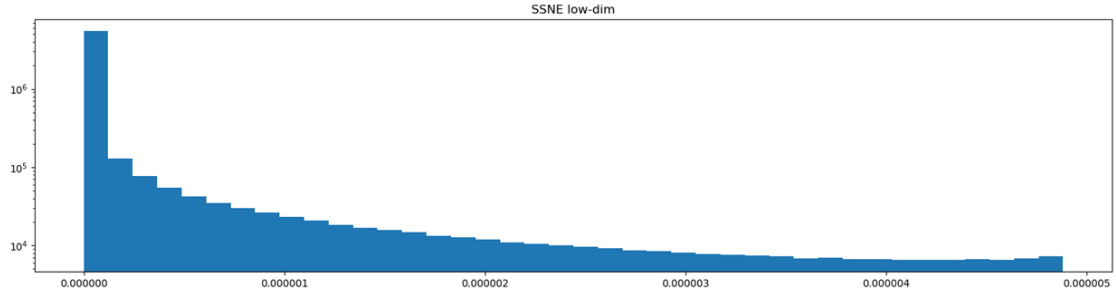
Upper gif: t-SNE scatter plot when doing iterate.
Lower gif: SSNE scatter plot when doing iterate.

Visualize the distribution of pairwise similarities in both high dimensional space and low-dimensional space.



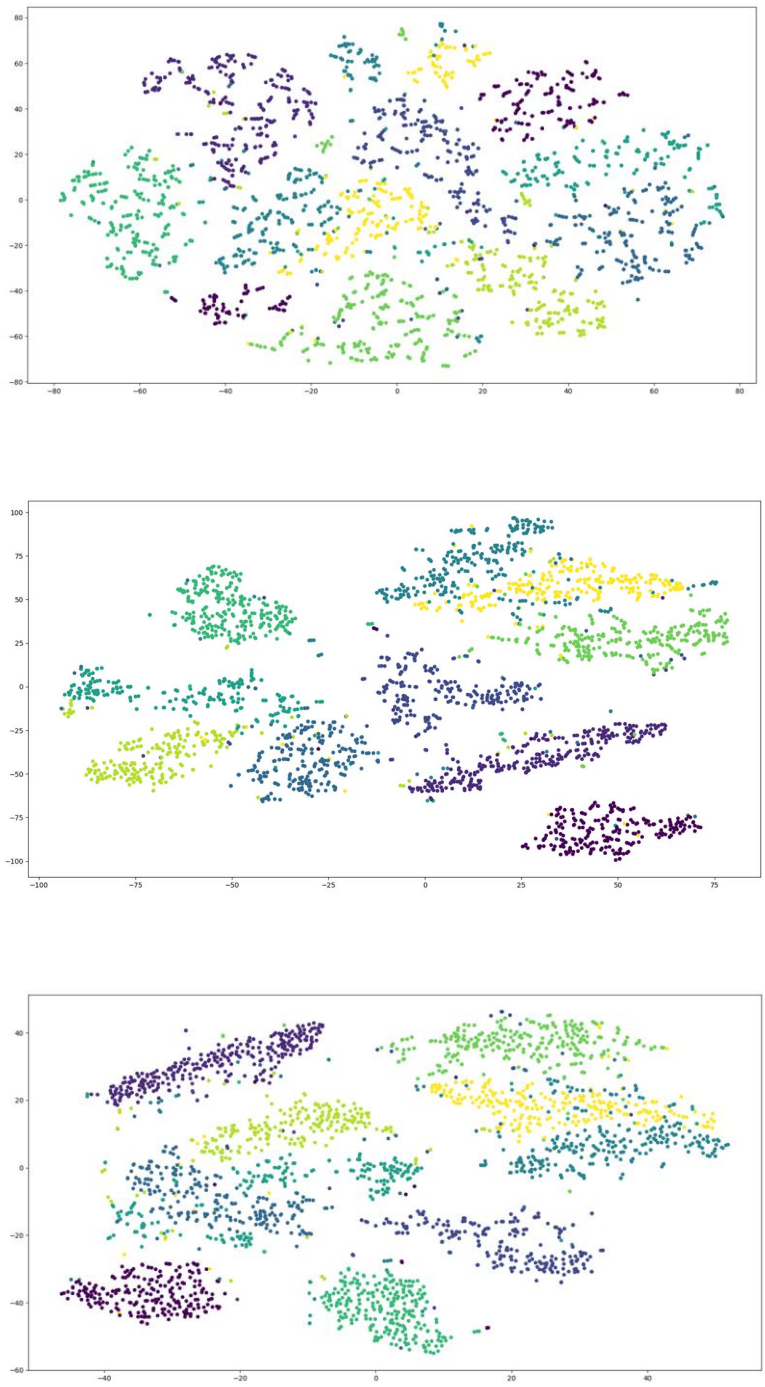




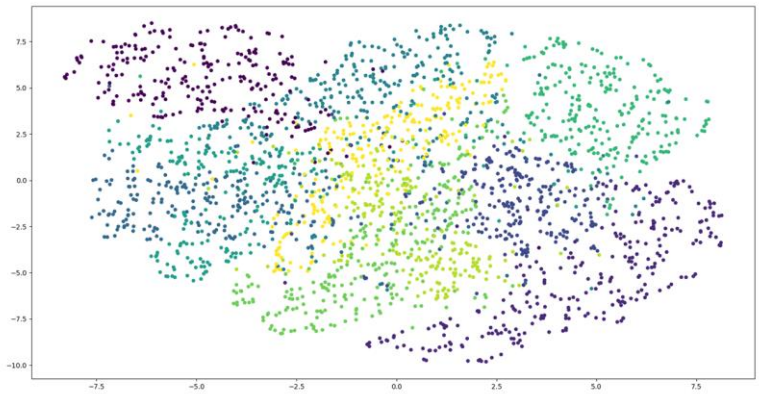


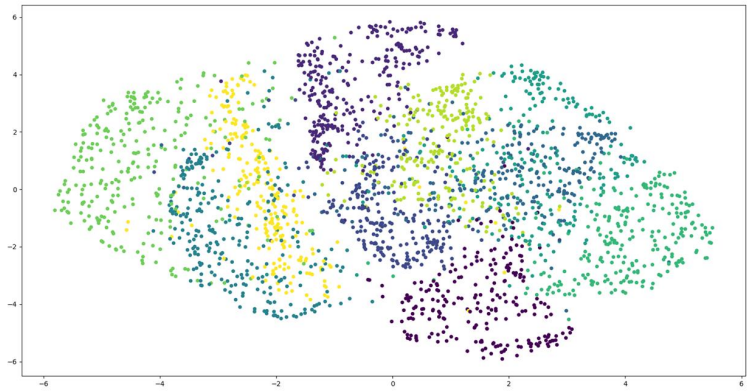
Both tSNE & SSNE case are converge after 500 iterations.
We can found that in t-SNE，the pairwise similarity in the low-dim is in a range from 0 to 0.00006，but in SSNE，the pairwise similarity in the low-dim is only in a range from 0 to 0.000005，which is a much smaller range comparing with t-SNE.
Therefore may cause the crowded problem.

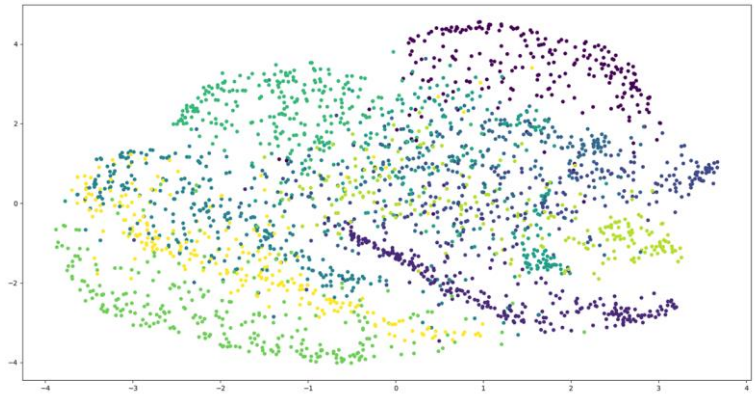
Changing the perplexity of t-SNE and SSNE.



From top to down: t-SNE with perplexity 5, perplexity 20, perplexity 50







From top to down: SSNE with perplexity 5, perplexity 20, perplexity 50

We can found that with higher perplexity，the global structure is more obvious，but it may be impossible to distinguish between groups. In lower perplexity，only a few neighbors are influential and may split the same group into multiple groups.