```
import numpy as np
import sklearn
from sklearn.model_selection import train_test_split
from sklearn import preprocessing
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import classification_report
from sklearn.metrics import accuracy_score
from sklearn.metrics import confusion_matrix, RocCurveDisplay,
roc_curve
from sklearn.metrics import ConfusionMatrixDisplay, confusion_matrix
import matplotlib.pyplot as plt

[X, y, name]=np.load("TP1.npy", allow_pickle=True)

print("X.shape = ", X.shape)
print("y.shape = ", y.shape) # 7 classes
print("name.shape = ", name.shape) # 7 noms

#on affiche une image et son label
plt.imshow(X[0].reshape(62,47), cmap='gray')
```

```
X.shape =  (1288, 2914)
y.shape =  (1288,)
name.shape =  (7,)
```

```
<matplotlib.image.AxesImage at 0x7f929a84f220>
```

Sachant que X resprésente les features, y les labels et name le nom des classes, déterminer la taille des images , le nombre d'images et le nombre de classes.

- il y au total 1288 images dans la base de données
- chaque image a 2914 pixels
- il y a 7 personnes = 7 classes
- vecteur avec [[image1], [image2], ... , [image1288]]

Partitionner la base en une base d'apprentissage et une base de test en mettant 25% des données en test (fonction train_test_split()) pour obtenir les variables X_train, X_test, y_train et y_test

```
# cela signifier que 25% des données seront utilisées pour le test et
75% pour l'entrainement

X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.25)

print("taille base de train = ", X_train.shape)
print("taille base de test = ", X_test.shape)

taille base de train =  (966, 2914)
taille base de test =  (322, 2914)
```

Mettre en forme les données (train et test) en utilisant la fonction classe StandardScaler.

Question : A quoi sert cette fonction, en quoi consiste la mise en forme des données ?

--> StandardScaler permet de centrer et réduire les données. Cela permet d'éviter que certaines features aient plus d'importance que d'autres. Sert a uniformiser les données en les mettant sur une même échelle. Cela permet de faciliter la comparaison entre les données.

Attention : il ne faut pas scale 2 moyennes base de test et base de train : il faut calculer moyenne et ecart type sur la base de train, et ensuite scaler toutes les données avec cela

- On fit le scaler sur les données d'entrainement
- Puis on `scale.transfrom` les données d'entrainement et de test avec ce scaler

```
#données train
scaler = preprocessing.StandardScaler().fit(X_train)
print("AVANT : moyenne = ", np.mean(X_train), " et ecart-type = ",
np.std(X_train))

#permet de centrer et réduire les données
X_train = scaler.transform(X_train)
X_test = scaler.transform(X_test)
```

```
print("APRES : moyenne = ", np.mean(X_train), " et ecart-type = ",
np.std(X_train))
```

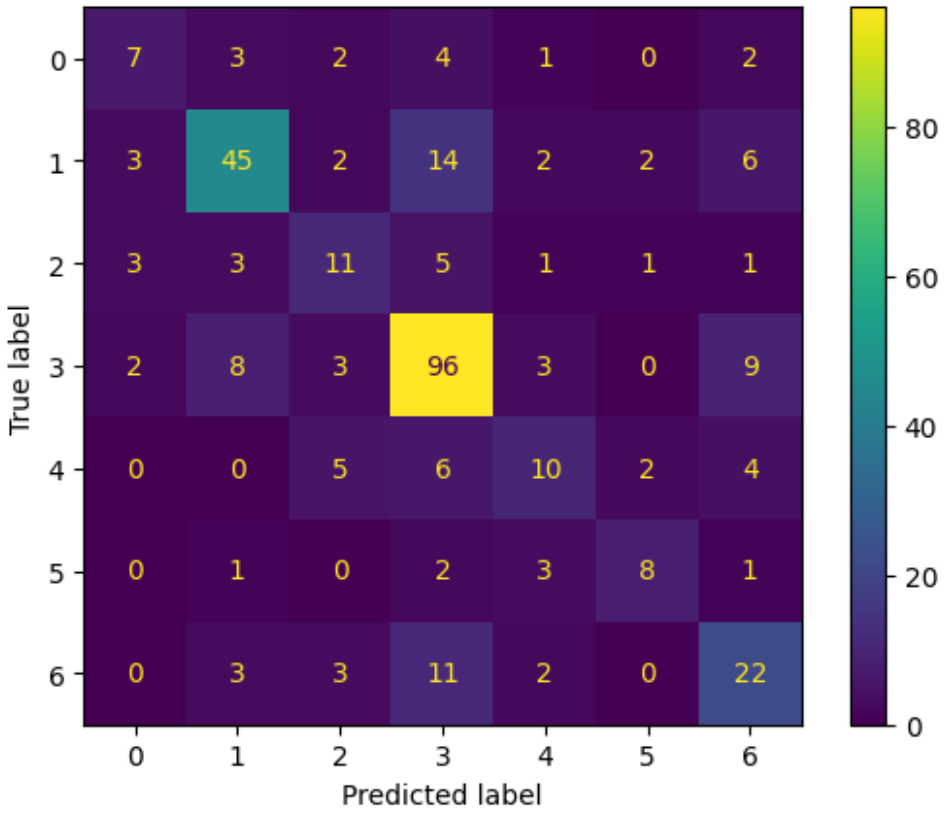
```
AVANT : moyenne =  132.13582  et ecart-type =  44.379486
APRES : moyenne =  5.4206756e-10  et ecart-type =  0.9999999
```

Classification kkp

```
#classification un 1-kppv

print("distance euclidienne p=2")
classifieur = KNeighborsClassifier(n_neighbors=1,p=2)
classifieur.fit(X_train,y_train)

predicted = classifieur.predict(X_test)

print("Le taux du classifieur est de " , accuracy_score(y_test,
predicted)*100, "%")
```

```
distance euclidienne p=2
Le taux du classifieur est de  61.80124223602485 %
```

```
matrice_confusion = confusion_matrix(y_test, predicted)
cm_display = ConfusionMatrixDisplay(matrice_confusion).plot()
```



Questions

- Que représente la matrice de confusion ?
- Que vaut sa somme ? -Est-ce que les classes sont équilibrées ?
- Que représente le rapport de classification ? Retrouver chacun de ses éléments à partir de la matrice de confusion

1) La matrice de confusion est de tailles 7*7 et représente les données après classification de la base de test.

2) Les classes ne sont pas équilibrés : on voit que la classe 3 est surepresenté

Retrouver les elements :

```
somme = np.sum(matrice_confusion)
print("le nombre d'element de la base de donnée de test de ", somme)
```

```
le nombre d'element de la base de donnée de test de  322
```

```
#rapport de classification
target_names = ['class 0', 'class 1', 'class 2', 'class 3','class
4','class 5','class 6']
print(classification_report(y_test, predicted,
target_names=target_names))
```

```
              precision    recall  f1-score   support

     class 0       0.47      0.37      0.41        19
     class 1       0.71      0.61      0.66        74
     class 2       0.42      0.44      0.43        25
     class 3       0.70      0.79      0.74       121
     class 4       0.45      0.37      0.41        27
     class 5       0.62      0.53      0.57        15
     class 6       0.49      0.54      0.51        41

    accuracy                           0.62       322
   macro avg       0.55      0.52      0.53       322
weighted avg       0.61      0.62      0.61       322
```

```
# pour retrouver la précision de la classe 0
#Percentage of correct positive predictions relative to total positive
predictions.

nombre_exemple_classe_0 = np.sum(matrice_confusion[0])
print("precision de la classe 0 = ", matrice_confusion[0]
[0]/nombre_exemple_classe_0)
```

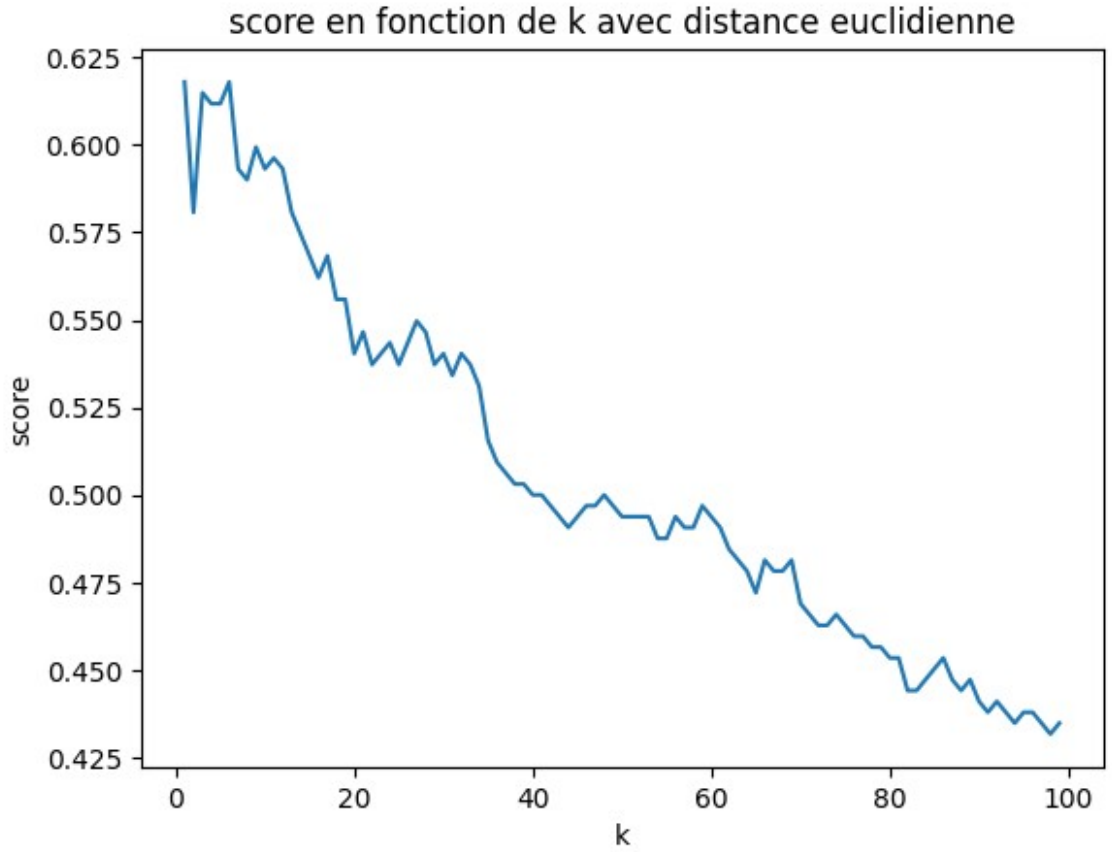
```
precision de la classe 0 =  0.3684210526315789
```

précision : % de True Positive qui est correct par rapport a toutes les predicitions

```
best_k = 0
max_score = 0
tab_score=[]
 #de 1 a 15
k_neighboors= np.array(range(1,100))

for k in k_neighboors:
    knn = KNeighborsClassifier(n_neighbors=k)
    knn.fit(X_train, y_train)
    predicted = knn.predict(X_test)
    score =  accuracy_score(y_test, predicted)
    if score> max_score:
        max_score = accuracy_score(y_test, predicted)
        best_k = k

    tab_score.append(score)

print("============================================================")
print("Le meilleur k est ", best_k, " avec un score de ",
max_score*100, "%")
print("============================================================")
```

```
============================================================
Le meilleur k est  1  avec un score de  61.80124223602485 %
============================================================
```

```
#tracé des courbes

import matplotlib.pyplot as plt
plt.plot(k_neighboors, tab_score)
plt.xlabel("k")
plt.ylabel("score")
plt.title("score en fonction de k avec distance euclidienne")
plt.show()
```

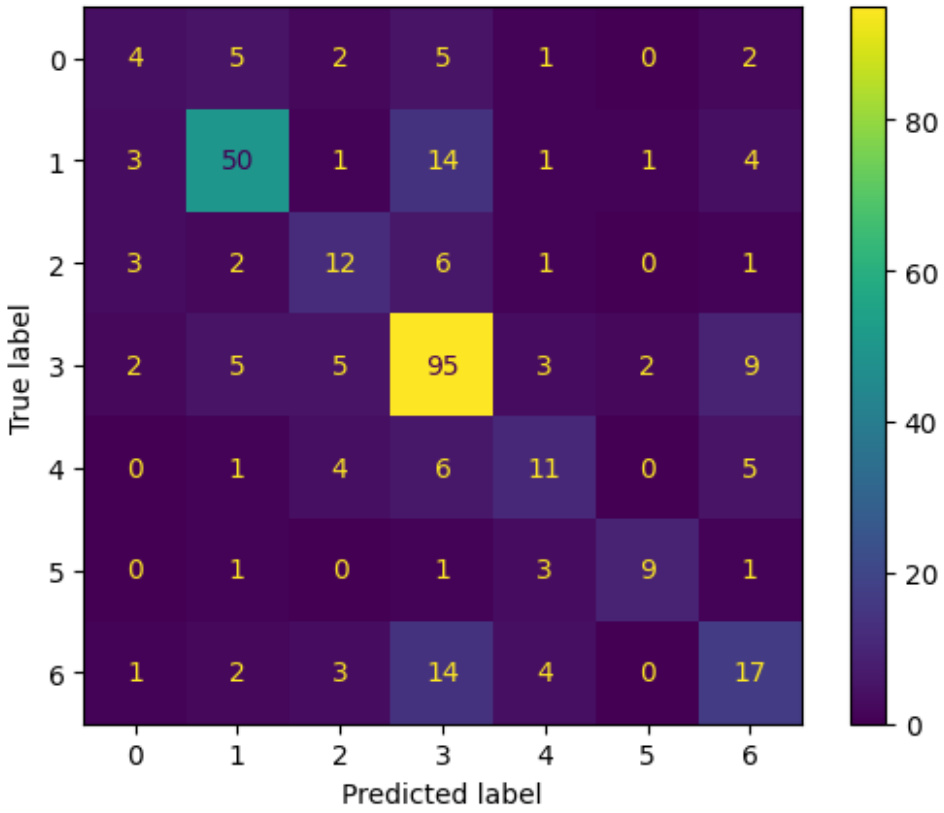


avec la distance de Manhattan

```
#classification un 1-kppv

print("distance Manhattan p=1")
classifieur = KNeighborsClassifier(n_neighbors=1,p=1)
classifieur.fit(X_train,y_train)
```

```
#predict exemple de test
predicted = classifieur.predict(X_test)


print("Le taux du classifieur est de " , accuracy_score(y_test,
predicted)*100, "%")

#matrice de confusion
from sklearn.metrics import confusion_matrix, RocCurveDisplay,
roc_curve
from sklearn.metrics import ConfusionMatrixDisplay, confusion_matrix

matrice_confusion = confusion_matrix(y_test, predicted)

cm_display = ConfusionMatrixDisplay(matrice_confusion).plot()
```

```
distance Manhattan p=1
Le taux du classifieur est de  61.49068322981367 %
```




```
best_k = 0
max_score = 0
tab_score=[]
 #de 1 a 15
```

```
k_neighboors= np.array(range(1,100))

for k in k_neighboors:
    knn = KNeighborsClassifier(n_neighbors=k,p=1)
    knn.fit(X_train, y_train)
    predicted = knn.predict(X_test)
    score =  accuracy_score(y_test, predicted)
    if score> max_score:
        max_score = accuracy_score(y_test, predicted)
        best_k = k

    tab_score.append(score)

print("=============================================================")
print("Le meilleur k est ", best_k, " avec un score de ",
max_score*100, "%")
print("=============================================================")


plt.plot(k_neighboors, tab_score)
plt.xlabel("k")
plt.ylabel("score")
plt.title("score en fonction de k avec distance euclidienne")
plt.show()
```

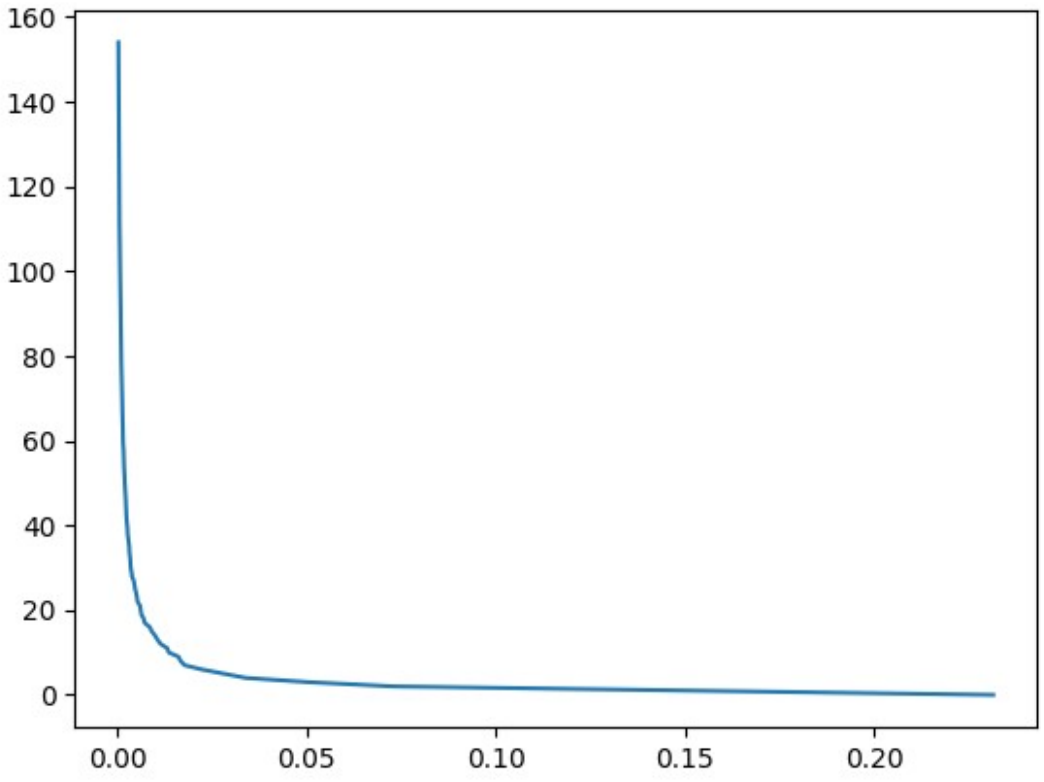
Analyse en composantes principales et classification

methode : on commence avec n_components assez grand, et on recupere les valeurs propres. On trace ensuite en fonctions des dimensions.

```
from sklearn.decomposition import PCA


#on doit mettre 966 max car on a 966 features
pca = PCA(n_components=966)
pca.fit(X_train) #juste sur la base de train
#on ne peut pas mettre plus que le nombre de features.
#
#Percentage of variance explained by each of the selected components.
valeur_propre = pca.explained_variance_ratio_
# valeurs propres normalisées a 1 : somme des valeurs propres = 1
#on cherche donc les n premieres valeurs propres qui expliquent 95% de
la variance
print("somme des valeurs propres = ", np.sum(valeur_propre))

n=0
```

```
while np.sum(valeur_propre[0:n])<0.95:
    n+=1
print("nombre de valeurs propres pour expliquer 95% de la variance =
", n)



#tracé des valeurs propres en fonction des domensios
plt.plot(valeur_propre[0:n], np.arange(0,n))
```

```
somme des valeurs propres =  1.0
nombre de valeurs propres pour expliquer 95% de la variance =  155

[<matplotlib.lines.Line2D at 0x7f92a195fa00>]
```



#on refais la classification en prenant compte la reduction de dimensions

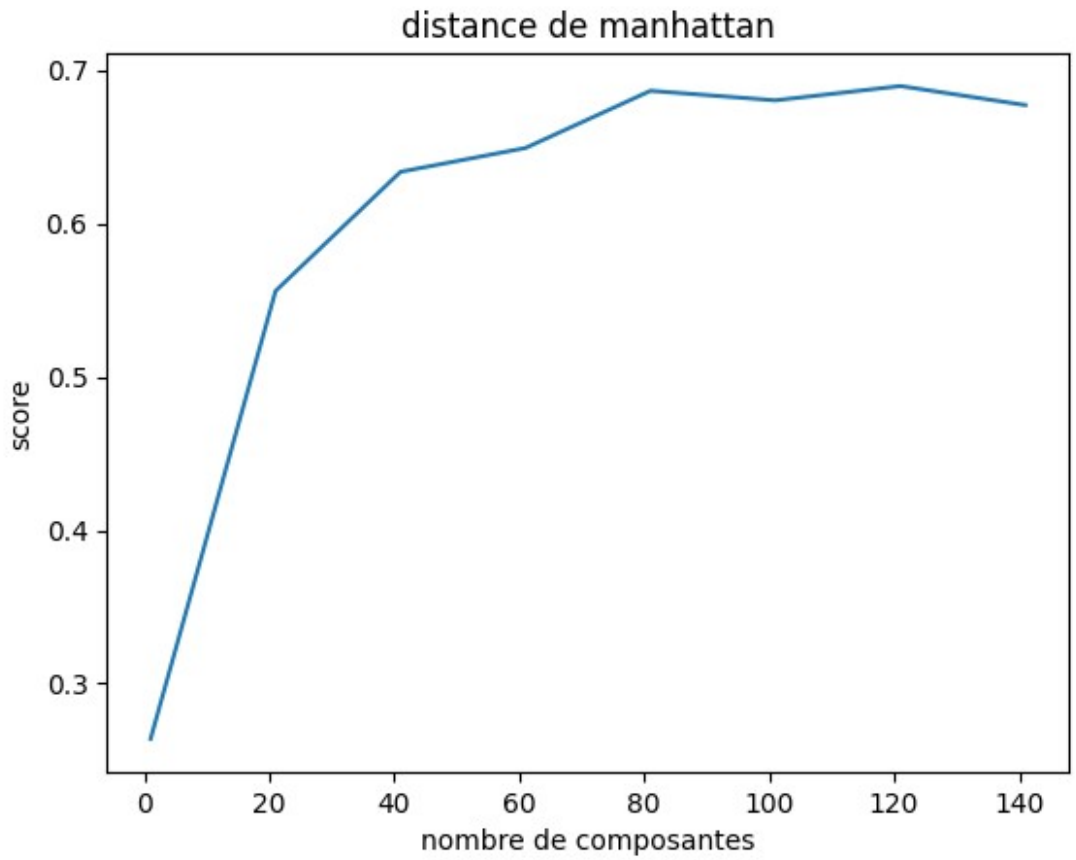
```
pca = PCA(n_components=n)
pca.fit(X_train)


X_train_pca = pca.transform(X_train)
X_test_pca = pca.transform(X_test)

# PCA nous permet de arde n composantes qui expliquent 95% de la
variance, ie n pixels par image
```

```
print("taille base de train | AVEC PCA ", X_train_pca.shape, "SANS PCA
= ", X_train.shape)


knn = KNeighborsClassifier(n_neighbors=1,p=1)
knn.fit(X_train_pca, y_train)
predicted = knn.predict(X_test_pca)
score =  accuracy_score(y_test, predicted)
print("Le taux du classifieur est de " , score*100, "%")
```

```
taille base de train | AVEC PCA  (966, 155) SANS PCA =  (966, 2914)
Le taux du classifieur est de  70.1863354037267 %
```

```
# tracé du score en fonction du nombre de composantes

components = np.array(range(1,n,20))
tab_score = []

for n_components in components:
    pca = PCA(n_components=n_components)
    pca.fit(X_train)
    X_train_pca = pca.transform(X_train)
    X_test_pca = pca.transform(X_test)

    knn = KNeighborsClassifier(n_neighbors=1,p=1)
    knn.fit(X_train_pca, y_train)
    predicted = knn.predict(X_test_pca)
    score =  accuracy_score(y_test, predicted)
    tab_score.append(score)

plt.plot(components, tab_score)
plt.title("distance de manhattan")
plt.xlabel("nombre de composantes")
plt.ylabel("score")
```

```
Text(0, 0.5, 'score')
```



conclusion : plus on garde un nombre important de composants, plus le score est elevée car on a plus de dimensions

Questions

- Que représentent les valeurs renvoyées par pca.explained_variance_ratio_ ?

pca.explained_variance_ratio_ renvoie les variances pour chaque axe.

- Combien de composantes sont nécessaire pour avoir une bonne classification

il faut garder les n premieres composantes qui assurent que l'on garde au minimum 0.9 % de la variance (information)

- Comment varient les temps de calcul en fonction du nombre de composantes ? plus il y a de dimensions, plus le temps de calcul monte

V. Analyse en composantes principales et reconstruction

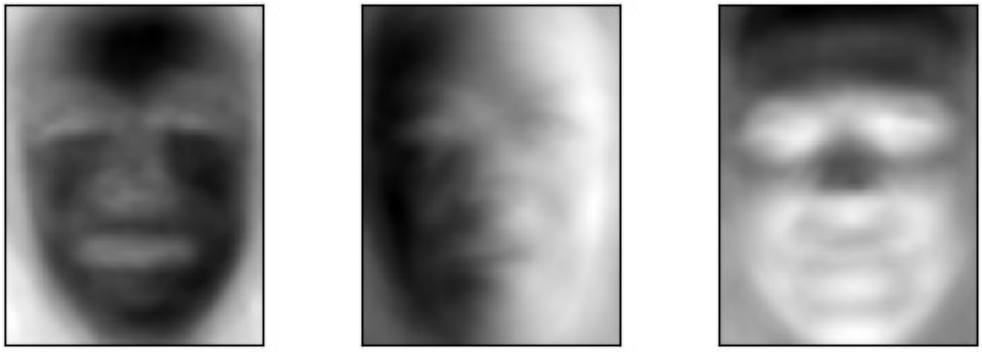
Définissez la décomposition en utilisant la fonction PCA() en conservant 50 composantes et l'appliquer sur les données en utilisant la méthode fit(). Récupérer les vecteurs propres en

utilisant une méthode de PCA(). Redimensionner les vecteurs propres en images propres (np.reshape()) de manière à pourvoir les visualiser sous forme d'images (array de taille 50x62x47). On utilisera la fonction plot_gallery() pour la visualisation.

```
pca = PCA(n_components=50)
pca.fit(X_train)
X_train_pca = pca.transform(X_train)
X_test_pca = pca.transform(X_test)

knn = KNeighborsClassifier(n_neighbors=1,p=1)
knn.fit(X_train_pca, y_train)


eigenvalues = pca.components_

print("eigenvalues.shape = ", eigenvalues.shape)

eigenvalues = np.reshape(eigenvalues, (50,62,47))

print("eigenvalues.shape = ", eigenvalues.shape) # 50 images de 62x47
```

```
eigenvalues.shape =  (50, 2914)
eigenvalues.shape =  (50, 62, 47)
```

```
def plot_gallery(images):
     #    Affiche les 12 premières images contenues dans images
     #    images est de taille Nb image*Ny*Nx
    plt.figure(figsize=(7.2, 7.2))
    plt.subplots_adjust(bottom=0, left=.01, right=.99, top=.90,
hspace=.35)
    for i in range(3):
        plt.subplot(3, 4, i + 1)
        plt.imshow(images[i], cmap=plt.cm.gray)
        plt.xticks(())
        plt.yticks(())
    plt.show()


plot_gallery(eigenvalues)
```



Les vecteurs propres sont de dimension (1,2914). Il y en a 50 (50 axes ou il y a le plus de variance des données). Ils représentent les axes principaux de la base de données.

on a gardé donc 50 vecteurs propres de taille (1,2914) : chaque image est une combinaison linéaire de ces 50 vecteurs propres, representée par par un vecteur de taille 50.

[x1, ..., x50] : chaque xi est un coefficient de combinaison linéaire

Pour reconstruire les images de test, on utilise la fonction inverse_transform() de la classe PCA.

```
#methode poour reconstruire les images

[X, y, name]=np.load("TP1.npy", allow_pickle=True)
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.25)


print("########## IMGAES ORIGINALES ##########")
plot_gallery(X_train[0:12].reshape(12,62,47))


#scale pca puis inverse pca
scaler = preprocessing.StandardScaler().fit(X_train)
X_train = scaler.transform(X_train)
X_test = scaler.transform(X_test)

print("########## IMGAES CENTREES REDUITES ##########")
plot_gallery(X_train[0:12].reshape(12,62,47))


#pca : on garde 50 composantes
pca = PCA(n_components=50)
pca.fit(X_train)
X_train_pca = pca.transform(X_train) # cette commande permet de
projeter les données sur les 50 composantes principales
X_test_pca = pca.transform(X_test)
print("X_train_pca.shape = ", X_train_pca.shape) #pour les 966 images,
on a un vecteur de 50 composantes tq
```

```
# image = X_train_pca[0]*eigenvalues[0] +
X_train_pca[1]*eigenvalues[1] + ... + X_train_pca[49]*eigenvalues[49]


#pour visualiser les 50 composantes principales
eigenvalues = pca.components_
eigenvalues = np.reshape(eigenvalues, (50,62,47))
print("########## 3 COMPOSANTES PRINCIPALES ##########")
plot_gallery(eigenvalues)


#on affiche la projection des 12 premières images de la base de train
sur les 50 composantes principales

X_test_reconstruit = pca.inverse_transform(X_test_pca)
print("############# APRES PCA INVERSE #############")
plot_gallery(X_test_reconstruit[0:12].reshape(12,62,47))

#de scale
print("############# APRES CENTRAGE ET REDUCTION INVERSE
#############")
X_test_reconstruit_descale =
scaler.inverse_transform(X_test_reconstruit)
plot_gallery(X_test_reconstruit_descale[0:12].reshape(12,62,47))

########## IMGAES ORIGINALES ##########
```

```
########## IMGAES CENTREES REDUITES ##########
```

```
X_train_pca.shape =  (966, 50)
########## 3 COMPOSANTES PRINCIPALES ##########
```

```
############# APRES PCA INVERSE #############
```

```
############# APRES CENTRAGE ET REDUCTION INVERSE #############
```

```
taux_compression = 1 - (50/2914)
print("taux de compression = ", taux_compression*100, "%")
```

```
taux de compression =  98.28414550446122 %
```

Faire varier le nombre de composantes conservées et calculer l'erreur de reconstruction (norme L2). Afficher l'erreur de reconstruction en fonction du nombre de composantes.

```
[X, y, name]=np.load("TP1.npy", allow_pickle=True)

scaler = preprocessing.StandardScaler().fit(X_train)
X_train = scaler.transform(X_train)
X_test = scaler.transform(X_test)

#pca

pca = PCA(n_components=966)
pca.fit(X_train)
X_train_pca = pca.transform(X_train)
X_test_pca = pca.transform(X_test)

X_test_reconstruit = pca.inverse_transform(X_test_pca)


X_test_reconstruit = scaler.inverse_transform(X_test_reconstruit)
plot_gallery(X_test_reconstruit[0:12].reshape(12,62,47))
```

```
composantes = np.array(range(1,150,10))
erreur_reconstruction = []

for nb_dimensions in composantes:
    #on fait la pca
    pca = PCA(n_components=nb_dimensions)
    pca.fit(X_train)
    X_test_pca = pca.transform(X_test)

    #on veut reconstruire chaque image de la base de test selon les 50
composantes principales
    #on va donc faire la transformée inverse de pca sur les 50
composantes principales
```

```
    X_test_reconstruit = pca.inverse_transform(X_test_pca)
    error = np.sum((X_test - X_test_reconstruit)**2) #erreur de toutes
les images

    #on scale
    error = np.sqrt(error/np.sum(X_test)) #erreur moyenne
    erreur_reconstruction.append(error)

plt.plot(composantes, erreur_reconstruction)


#reconstruction = decompression de nos images

#on fait la pca : on passe de 2000 dimensions a 50 : on a 50 vecteurs
propres
# chaque image est une combinaison lineaire de ces 50 vecteurs propres
# transformée inverse pour reussir a exprimer a1* V1 + a2*V2 + ... +
a50*V50
# on a une image reconstruite avec a nouveau 2000 dimensions =
décompression


 #principe
```

```
[<matplotlib.lines.Line2D at 0x7f92a71713d0>]
```

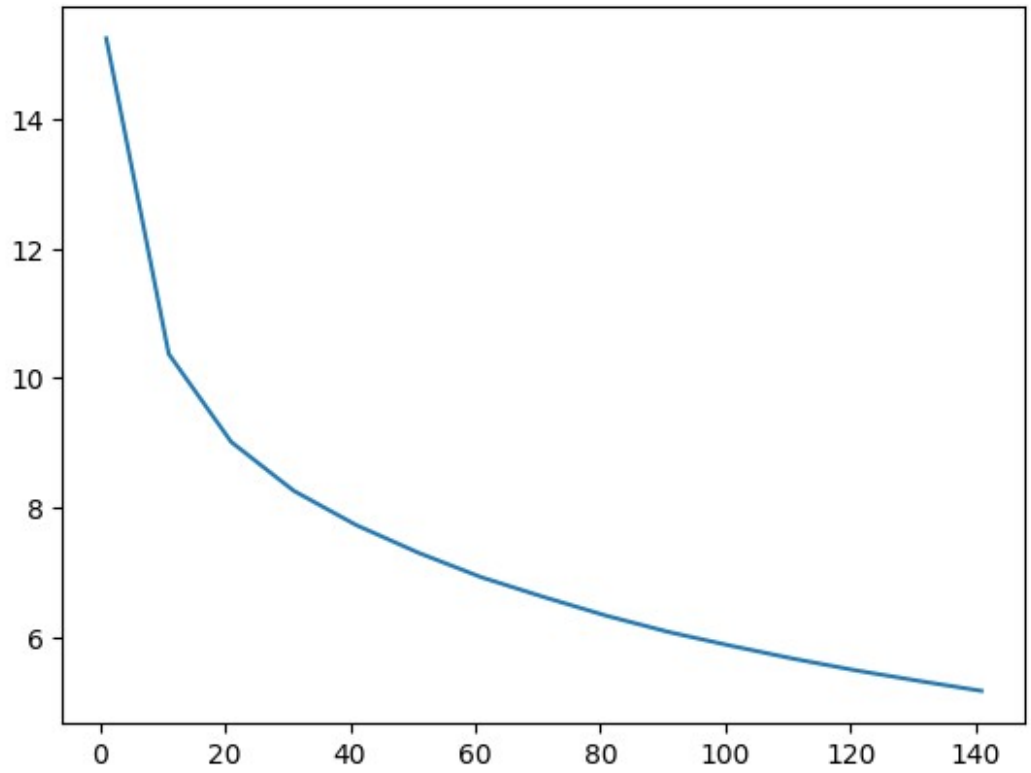


image cours :

Analyse en composante principale (ACP)

Si on ne conserve que les 5 premières dimensions, chaque visage de la base s'exprime comme une combinaison linéaire de ces 5 'eigen-image'

$$x_{i1}'\boldsymbol{i_1}' + x_{i2}'\boldsymbol{i_2}' + \ldots. + xi_5'\boldsymbol{i_5}'$$

Ainsi, tous les exemples sont représentés par un vecteur de dimension 5.

A partir de ce vecteur, on peut :

- Reconstruire les visages, on aura alors fait de la compression
- Reconnaitre les visages

Comment ?

Chaque visage est représenté uniquement par un vecteur de dimension 5 :

$$\boldsymbol{x_i}' = (x_{i1}'\ x_{i2}'\ \ldots\ xi_5')^T$$

- Pour le reconstruire, à la décompression, on utilise les eigen images

$$x_{i1}'\boldsymbol{i_1}' + x_{i2}'\boldsymbol{i_2}' + \ldots. + xi_5'\boldsymbol{i_5}'$$

- Pour le reconnaitre, on utilise le vecteur $\boldsymbol{x_i}' = (x_{i1}'\ x_{i2}'\ \ldots\ xi_5')^T$ comme codage du visage

Analyse en composante principale (ACP)

Si on ne conserve que les 5 premières dimensions, chaque visage de la base s'exprime comme une combinaison linéaire de ces 5 'eigen-image'

$$x_{i1}'\boldsymbol{i_1}' + x_{i2}'\boldsymbol{i_2}' + \ldots + xi_5'\boldsymbol{i_5}'$$

Ainsi, tous les exemples sont représentés par un vecteur de dimension 5.

A partir de ce vecteur, on peut :

- Reconstruire les visages, on aura alors fait de la compression
- Reconnaitre les visages

Comment ?

Chaque visage est représenté uniquement par un vecteur de dimension 5 :

$$\boldsymbol{x_i}' = (x_{i1}'\ x_{i2}'\ \ xi_5')^T$$

- Pour le reconstruire, à la décompression, on utilise les eigen images

$$x_{i1}'\boldsymbol{i_1}' + x_{i2}'\boldsymbol{i_2}' + \ldots + xi_5'\boldsymbol{i_5}'$$

- Pour le reconnaitre, on utilise le vecteur $\boldsymbol{x_i}' = (x_{i1}'\ x_{i2}'\ \ldots\ xi_5')^T$ comme codage du visage