Ce projet a pour but de vous faire découvrir une des premières techniques de reconnaissance faciale, celle utilisant les "eigenfaces".
Cette technique de reconnaissance utilise la méthode d'analyse en composantes principales (ACP) ou la méthode de décomposition en valeurs singulières (SVD).
L'idée centrale est de diminuer la dimension de l'espace de travail pour simplifier les données et leur interprétation.
Une fois cette dimension réduite, la comparaison d'un nouveau visage à ceux de la base de données devient moins coûteuse ce qui permet d'espèrer un bon taux de réussite.
Les "eigenfaces" sont obtenues lors d'une première phase d'apprentissage. Il s'agit tout simplement des composantes principales de la matrice regroupant les images des visages d'une partie de la base de
données.
Elles vont servir de base pour exprimer chacune des images. En se limitant aux premières composantes (une fraction qui représente un pourcentage suffisant de l'inertie totale) on réduit la dimension de représentation des images.
Ces composantes principales peuvent également être obtenues par la décomposition en valeurs singulières d'une matrice étroitement liée à celle de la base de données comme on va le voir dans la partie mathématique.

Outil Mathématique demandé

1. La norme subordonée matricielle $|||A|||_2 = \sqrt{\rho(AA^*)} = \sqrt{\rho(A^*A)}$

Si la matrice A est symétrique et réélle alors $|||A|||_2=\rho(A)$, ρ le rayon spectral de A

2. C est la matrice de covariance variance qui est une matrice symétrique définie positive et réelle, donc d'apès le théorème spectral C est diagonalisable

3. C est symétrique définie positive, donc ls valeurs propres de C sont positives

4. $C= Y^TY = (U\Sigma V^T)^T(U\Sigma V^T)= V\Sigma^TU^TU\Sigma V^T= V\Sigma^T\Sigma V^T = PDP^T$
car $U^TU=I_n$ var U est une matrice orthogonale
Donc par identification **P=V** et **D=**$\Sigma^T\Sigma$

5. Soit $(e_1,.....,e_k)$ une base orthonormée
Pour tout i $\neq k$ $<e_i , e_k> =0$ car la base est orthonormée
si i=k $<e_i , e_k> =1$
$x = \Sigma x_ie_i$, donc $<x,e_k> = <\Sigma x_ie_i , e_k> = \Sigma<x_ie_i , e_k> = \Sigma x_i<e_i , e_k>$
or pour tout i $\neq k$ $<e_i , e_k> =0$ $<x,e_k> = 0 + x_k<e_k,e_k> = x_k$
donc x_k peut s'écrire dans une base orthonormée comme $\mathbf{x_k = <x,e_k>}$

Pour avoir donc les lignes de Y dans une base orthonormée formée par P i**l faut multiplier Y par P.** On a donc **Z=YP**

Phase d'apprentissage

1. Pour la transformation de matrice en une ligne et une ligne de taille 4096 et
2. La lecture et transformation dans une matrice ligne de taille 4096 en une boucle avec python.
Avant tout on va importer les packages Numpy as np, matplotlib.pyplot as plt et os

On va parcourir les fichiers et récupérer les noms avec **os**

```
images = os.listdir("C:/Users/faous/Desktop/eisti/semestre 2/TP Analyse numérique/BDD1/FG1/")

X=[]
for i in range(0,100) :
    image=plt.imread("C:/Users/faous/Desktop/eisti/semestre 2/TP Analyse numérique/BDD1/FG1/" + images[i])
    image=np.resize(image,4096)
    X.append(image)
```

3. Normalisation de X et création de Y

```
"les moyennes des colonnes"
moyennes=np.mean(X,0)

"Les ecarts type des colonnes"
ecarts=np.std(X,0)

M=[]
M.append(moyennes)
M.append(ecarts)
```

Y=(X-moyennes)/ecarts

4. Décomposition en valeurs singulières

[U,S,Vt]=np.linalg.svd(Y,full_matrices= False)

S=np.diag(S) "Car S est une matrice diagonale avec sur la diagonales les valeurs propres de la matrice de covariance variance

"determination de P et D d'après la question 4 partie 3"
Avec numpy de Python
P=np.transpose(Vt)
D=np.dot(np.transpose(S),S) = S^TS

5. Choix du nombre k de composantes à retenir/colonnes à retenir (tester k entre 50 et 70)

"Inertie totale"
I=np.trace(D)

On teste des valeurs entre 50 et 70
D_k= D[:,0:k] $50 \leq k \leq 70$

I_k=np.trace(D_k)

proportion1=I_k/I

En se fixant un seuil a 95% On ne retiendra que les 55 premiières colonnes car pour les 55 premières colonnes on a une proportion 95%.

P_{55}=P[:,0:55]

6. Détermination de Z la matrice qui contient les cooordonées de toutes les images de la BDD par rapport aux composantes principales retenues

Z=np.dot(Y,P_{55})

Base de données en dimension réduite

On va maintenant exprimer toutes les images de la BDD dans la nouvelle base P_{55}
On refait la même chose que les questions 1. et 2. , mais cette fois ci pour 1680 images plutôt que 1000
X_m=[]
```
for i in range(0,1680) :
    image=plt.imread("C:/Users/faous/Desktop/eisti/semestre 2/TP Analyse
            numérique/BDD1/FG1/" + images[i])
    image=np.resize(image,4096)
```

X_m.append(image)

"Normalisation de Xm"
Y_m**=(X_m-M[0])/M[1]**

"Zm qui donne les coordonnées dans la base réduite"
Z=np.dot(Y_m,P_{55})

Reconnaissance faciale

1. Rechercher un visage inconnu dans BDD1 ou BDD2
1ère étape : On va pour cela construire la matrice X_2 de taille (100,4096) dont les 100 lignes dans la matrice represente les 100 images de BBD2

```
images2 = os.listdir("C:/Users/faous/Desktop/eisti/semestre 2/TP Analyse numérique/BDD2/FG2/")
"Construction d'une matirce formée d visages de la base de données BDD2"
X2=[]
for i in range(0,100) :
    image2=plt.imread("C:/Users/faous/Desktop/eisti/semestre 2/TP Analyse numérique/BDD2/FG2/" + images2[i])
    image2=np.resize(image2,4096)
    X2.append(image2)
```

2ème étape : Recherche d'un visage inconnu de BDD2 dans BBD1

On essaye de vérifier s'il y'a des images communes entre BDD1 et BDD2
Pour cela on écrit un algorithme qui compare chacune des 100 lignes de X_2 (100×4096) à toutes les 1680 lignes de la matrice X_m(1680×4096), c'est-à dire qui compare les cordonées de chacune des 100 images de BB2 aux coordonées des 1680 images de BDD1. Les lignes communes correspondent au visages communs.

Après application de l'algorithme, on remarque qu'il n' y a pas de visages communs, tous les visages de BDD2 sont différents des visages de BDD1.

On choisit donc une image quelconque de BDD2 pour la suite du traitement.

x_p= X_2[90].

2. Normalisation de cette image

On va utiliser la matrice M pour normaliser, on soustrait a la matrice x_p la première ligne de la Matrice M et on disvise par sa deuxième ligne.
Avec python on a:

y_p=(x_p-M[0])/-M[1]

3. Coordonnées de cette image dans la base des composantes principales retenues
Avec python :

Z_p=np.dot(y_p,P55)

4. Calcul des distances entre la nouvelle image et celles de la BDD avec Python:

```
di=[]
for i in range(0,1680):
    dist=zp-Z[i]
    d=np.linalg.norm(dist, 2)
    di.append(d)
```

Ce code cacule pour chaque i la différence dist=zp-Z[i] et calcule la norme matricielle 2 de dist pour chaque i et la stocke dans d_i .

5. Identification du visage par la recherche de la distance minimale
On utilise en python dist_min =np.min(d_i) pour avoir la distance minimale

Pour savoir a quelle visage cette image correspond il faut recupérer l'indice de l'image
indice_visage=np.argmin(di)

Le visage 90 de la BBD2 est proche au visage 626 de la BDD2 avec une distance minimale de 21
pour afficher ces images on a :

```
G=plt.imread("C:/Users/faous/Desktop/eisti/semestre 2/TP Analyse numérique/BDD2/FG2/"
+
images2[90])
plt.figure()
plt.imshow(G)

w=plt.imread("C:/Users/faous/Desktop/eisti/semestre 2/TP Analyse numérique/BDD1/FG1/" +
images[626])
plt.figure()
plt.imshow(w)
```

Travail de codage demandé

1. Fonction Apprentissage

```
def Apprentissage(images):


    X=[]
    path = os.listdir(images)
    for i in range(0,100):
        image=plt.imread(images + path[i])
        image=np.resize(image,4096)
        X.append(image)

    "Normalisation de X"

    "les moyennes des colonnes"
    moyennes=np.mean(X,0)

    "Les ecarts type des colonnes"
    ecarts_type=np.std(X,0)

    M=[]
    M.append(moyennes)
    M.append(ecarts_type)

    Y=(X-moyennes)/ecarts_type

    "Question 4"
    "Décomposition en valeurs singulières"

    [U,S,Vt]=np.linalg.svd(Y,full_matrices= False)

    S=np.diag(S)

    "determination de P et D d'après la question 4 partie 3"
    P=np.transpose(Vt)
    D=np.dot(np.transpose(S),S)

    "Question 5"
```

```
    "colonnes à retenir (tester k entre 50 et 70"

    "Inertie totale"
    I=np.trace(D)

    "on teste des valeurs entre 50 et 70"
    for i in range (50,71) :
        Dk=D[:,0:i]
        Ik=np.trace(Dk)
        proportion=Ik/I
        if proportion >=0.95 :
            k=i
            break
    P_k=P[:,0:k]
    """on fixe un seuil à 95% donc les k premières colonnes doivent expliquer au
     moins 95% de la variance. Mais on veut une réduction de dimensions, donc on
     prend le plus petit k qui donne une proportion >=95% """


    "détermination de Z"

    Z=np.dot(Y, P_k)

    return M, Z,  P_k
```

2. Fonction ajout

```
def ajout(image,M, P_k,Z):

    path = os.listdir(image)
    x_m=plt.imread(image + path[2])
    x_m=np.resize(x_m,4096)
    ym=(x_m-M[0])/M[1]
    z_m=np.dot(y_m,Pk)
    np.append(Z,z_m)
```

3. Fonction Reconnaissance

```
def reconaissance(image,M, P_k,Z,images):
```

```
path = os.listdir(image)
img=plt.imread(image + path[4])

x_m=np.resize(img,4096)
y_m=(x_m-M[0])/M[1]
z_m=np.dot(y_m,P_k)
d=[]
for x in Z:
    diff=z_m-x
    dist=np.linalg.norm(diff,2)
    d.append(dist)

min=np.argmin(d)

path = os.listdir(images)
imagess=plt.imread(images + path[min])

plt.figure()
plt.imshow(imagess)
plt.figure()
plt.imshow(img)

return min
```

Cette methode est efficace car elle permet de voir les images qui ont des pixels proches (les cordonnées proches) afin de faire la reconnaissance faciale
Il est nécessaire de définir un seuil de distance au dessus duquel on décide qu'in n'y a pas de ressemblance. En effet, en posant un seuil, on garantie un maximum de ressemblance et un minimum d'images similaires reconnues.