Rapport TS115 : Statistiques

16 avril 2021

HOULIER Thomas
BREJON Louis

ENSEIRB-MATMECA
Télécommunications - Groupe 10

Table des matières

Table des figures

Introduction

Dans ce TP, nous nous sommes intéressés à l'identification de visages en utilisant des outils d'apprentissage statistiques pour concevoir des algorithmes qui, à partir d'une base d'images de visages d'individus identifiés, est capable d'associer à une nouvelle image l'individu qui lui correspond. Ces algorithmes peuvent apprendre un modèle mathématique à partir de données disponibles, pour ensuite effectuer certaines tâches comme la classification, le clustering et la regression.
L'Analyse en Composantes Principales, notée ACP, est un outil classique d'analyse employé pour réaliser un apprentissage correct. Inventée par Karl Pearson en 1901, l'ACP a pour objectif d'effectuer une réduction de dimension des données de départ, en transformant un grand nombre de variables corrélées entre elles en un nombre réduit de variables décorrélées, appelées composantes principales, et décrivant de manière suffisante les données d'origine.

1 Principe de l'ACP

L'ACP consiste à projeter les données x_1, ..., x_n dans les directions orthogonales qui contiennent le plus d'« énergie » au sens statistique du terme, c'est-à-dire en terme de variance empirique. Plus formellement, on cherche une base de vecteurs orthonormés v_1, ..., $v_l \in \mathbb{R}^p$, tels que les variances empiriques E_j valent :

$$E_j = \frac{1}{n}\sum_{i=1}^{n} < v_j, x_i - \hat{x} >^2 \text{ tel que j } \in \text{ [1,l]} \tag{1}$$

1.1 Maximisation de l'énergie de projection : cas i = 1

Soit : $\hat{R} = \frac{1}{n}\sum_{i=1}^{n}(x_i - \hat{x})(x_i - \hat{x})^T$

On sait que : $\begin{cases} \hat{R} \text{ est définie semi-positive} \\ \lambda_1 \geq ... \geq \lambda_p \\ u_1, ..., u_p \text{ sont des valeurs propres orthonormées} \end{cases}$

Soit :

$$\begin{aligned} f(v) &= \frac{1}{n}\sum_{i=1}^{n} < v, x_i - \hat{x} >^2 \\ &= \frac{1}{n}\sum_{i=1}^{n} v^T(x_i - \hat{x})(x_i - \hat{x})^T v \\ &= v^T \hat{R} v \end{aligned}$$

Or, la matrice $\hat{R}$ est diagonalisable.
Donc, il existe une matrice D $\in \mathbb{D}_n$ et une matrice P $\in \mathbb{GL}_n$ tel que : $\hat{R} = P^T D P$

$$\mathrm{D} = \begin{pmatrix} \lambda_1 & 0 & 0 \\ 0 & .. & 0 \\ 0 & 0 & \lambda_p \end{pmatrix} \text{ et } \mathrm{P} = \begin{pmatrix} u_{11} & u_{12} & .. & u_{1p} \\ u_{21} & u_{22} & .. & u_{2p} \\ .. & .. & .. & .. \\ u_{p1} & u_{p2} & .. & u_{pp} \end{pmatrix}$$

Donc

$$f(v) = v^T (\sum_{i=1}^{p} \lambda_i u_i u_i^T) v$$

On obtient donc l'équation suivante avec comme condition d'égalité $v = u_1$.

$$\sum_{i=1}^{p} \lambda_i < v, u_i >^2 \geq \lambda_1 \tag{2}$$

En effet, $\lambda_1 \leq ... \leq \lambda_p$ et ||v||=1 On obtient donc :

$$E_1 = \lambda_1 \tag{3}$$

1.2 Maximisation de l'énergie de projection : cas i ≥ 2

Pour étendre le résultat vu Partie 1.2, on reprend le calcul précédent et on utilise la propriété d'orthogonalité suivante : $v \perp u_1 \perp ... \perp u_{j-1}$. Reprenons l'équation (2) transformée pour correspondre aux hypothèses de cette question :

$$\sum_{i=1}^{p} \lambda_i < v, u_i >^2 \geq \lambda_j$$

En effet, le produit scalaire $< v, u_i >$ sera nul pour tous les termes de la somme excepté pour le vecteur u_j qui n'est pas orthogonal au vecteur v.

De la même manière, nous obtenons :

$$E_j = \lambda_j \tag{4}$$

1.3 Expression de $\kappa(l)$

Afin de réduire les temps de calcul et la complexité calculatoire de l'implémentation du code, nous allons utiliser **l** composantes principales correspondants au sous espace de projection $S = Vect(u_1, ..., u_l)$. Dans cette question, nous allons exprimer le ratio de l'énergie de projection sur l'énergie totale associée aux données, $\kappa(l)$. Dans un premier temps, la projection orthognale sur S se note :

$$\Pi_S(x) = \sum_{j=1}^{l} < u_j, x > u_j \tag{5}$$

Dans un second temps, le ratio $\kappa(l)$ s'exprime :

$$\begin{aligned}
\kappa(l) &= \frac{\frac{1}{n}\sum_{i=1}^{n}||\Pi_S(x_i-\overline{x})||_2^2}{\frac{1}{n}\sum_{i=1}^{n}||x_i-\overline{x}||_2^2} \\
&= \frac{\frac{1}{n}\sum_{i=1}^{n}||\sum_{k=1}^{l}<x_i-\overline{x},u_k>||_2^2}{\frac{1}{n}\sum_{i=1}^{n}||x_i-\overline{x}||_2^2} \\
&= \frac{\sum_{k=1}^{l}\frac{1}{n}\sum_{i=1}^{n}<x_i-\overline{x},u_k>^2}{\frac{1}{n}\sum_{i=1}^{n}||x_i-\overline{x}||_2^2} \\
&= \frac{\frac{1}{n}\sum_{k=1}^{l}u_k^T R u_k}{\frac{1}{n}\sum_{i=1}^{n}||x_i-\overline{x}||_2^2} \\
&= \frac{\frac{1}{n}\sum_{k=1}^{l}\lambda_k}{\frac{1}{n}\sum_{i=1}^{n}||x_i-\overline{x}||_2^2} \\
&= \frac{\sum_{k=1}^{l}\lambda_k}{\sum_{i=1}^{n}\lambda_i}
\end{aligned}$$

1.4 Méthode de Gram

La méthode de Gram permet de réduire le coût calculatoire de la décomposition spectrale de $\hat{R}$. Pour la suite, nous supposerons que x_1, ..., x_n sont linéairement indépendants pour simplifier les développemenets tel que :

$$\hat{R} = XX^T \text{ avec } X = \frac{1}{\sqrt{n}}[x_1-\hat{x},...,x_n-\hat{x}] \tag{6}$$

1.4.1 Matrice $\mathbf{A = X^T X}$ définie positive

Soit $\begin{cases} X^TX \text{ la matrice de Gram} \\ v \text{ est un vecteur propre de la matrice de Gram} \\ u \text{ est un vecteur propre de } XX^T \text{ tel que } u = Xv \end{cases}$

Alors,

$$\begin{aligned}
(XX^T)u &= (XX^T)(Xv) \\
&= X[X^TXv] \\
&= X[\lambda v] \\
&= \lambda Xv \\
&= \lambda u
\end{aligned}$$

Donc $\boxed{\text{u est un vecteur propre de } XX^T \text{ associé à la même valeur propre}}$

De plus, toutes les valeurs propres (λ_i) sont positives donc $\boxed{X^TX \text{ est définie positive}}$ d'après le résultat obtenu précédemment.

1.4.2 Orthonormalisation de la matrice U

Soit x_1, ..., x_n un ensemble de vecteurs propres orthonormés de $X^T X$ tel que $\begin{cases} V = [v_1, ..., v_n] \\ U = XV[(XV)^T XV]^{-1/2} \end{cases}$

Alors,

$$\begin{aligned} UU^T &= XV[(XV)^T XV]^{-1/2}(XV[(XV)^T XV]^{-1/2})^T \\ &= XV[(XV)^T XV]^{-1/2}[(XV)^T XV]^{-1/2}(XV)^T \\ &= XV[(XV)^T XV]^{-1}(XV)^T \\ &= (XV)[(XV)^{-1}((XV)^T)^{-1}](XV)^T \\ &= (XV)(XV)^{-1}((XV)^T)^{-1}(XV)^T \\ &= I.I \\ &= I \end{aligned}$$

Donc la matrice U est une matrice orthogonale à colonnes orthonormées de vecteurs propres de $\hat{R}$ associés aux valeurs propres non nulles d'après les résultats précédents.

2 Réduction de dimension et eigenfaces

Dans cette partie, nous allons calculer les composantes principales d'une base de données d'images contenant des visages humains. Pour cela, nous utilisons les résultats vus Partie 1.4.1 et 1.4.2 afin de réduire les temps de calcul d'un tel programme.

La base d'apprentissage qui sera utilisé dans cette partie est illustrée ci-dessous :

FIGURE 1 – Base d'apprentissage

2.1 Représentation des n eigenfaces

Ainsi, on utilise la matrice de Gram aux propriétés intéressantes afin d'accéder aux valeurs propres et vecteurs propres de la base de données. Les **eigenfaces** correspondent aux vecteurs propres contenus dans la matrice U. Après implémentation sous Matlab, nous obtenons les **eigenfaces** suivantes :

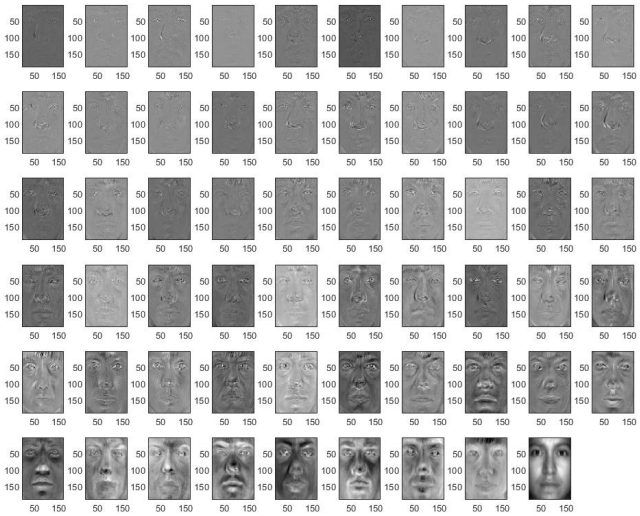

FIGURE 2 – Eigenfaces

On remarque que les eigenfaces associées à des valeurs propres plus grandes, sont plus marquées : elles auront un impact plus fort lors de la reconnaissance faciale.

2.2 Evolution de l'image reconstruite après ACP en fonction de l

Pour reconstruire une image à l'aide des composantes principales obtenues précédemment (eigenfaces), on projette orthogonalement le vecteur dans le sous espace de projection $S = Vect(u_1, ..., u_l)$. L'implémentation de cette partie est faites dans le fichier *Proj.m*. Le paramètre **l** est choisi manuellement dans cette partie. Par exemple, pour l=60, on obtient l'image reconstituée suivante à partir du vecteur X :

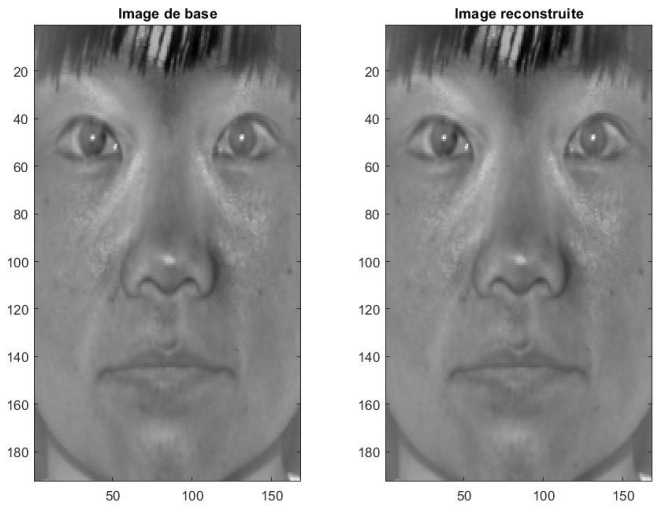

FIGURE 3 – Image reconstruite après la projection sur la base de taille l=59

Ce résultat est rassurant puisque plus l se rapproche de 60, plus la reconstruction doit être proche de la réalité. Pour observer l'influence du paramètre l sur la qualité de reconstruction d'une image, nous avons affiché l'évolution de la reconstruction en fonction du paramètre l :

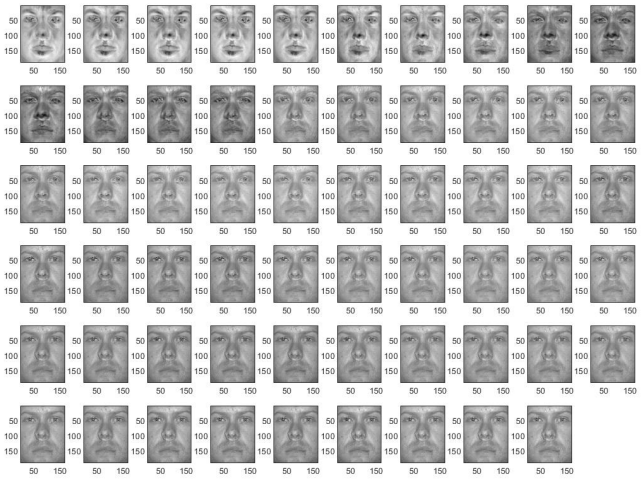

FIGURE 4 – Evolution de la reconstruction en fonction du paramètre l

On remarque bien que la reconstruction s'améliore au fur et à mesure que l augmente (l=59 correspondant à l'image en bas à droite).

2.3 Représentation du ratio de reconstruction $\kappa(l)$

Afin d'optimiser le modèle et les performances du programme, nous devons choisir le paramètre l selon le critère suivant :

$$l_* = min[l : \kappa(l) \geq \alpha]$$

où α est compris entre 0 et 1. Il sera fixé à 0,9 dans la suite.
Sous Matlab, la détermination du paramètre optimal l_* se fait dans le fichier *crit_K.m*. En traçant l'évolution du ratio κ en fonction du paramètre l, on obtient la figure suivante :

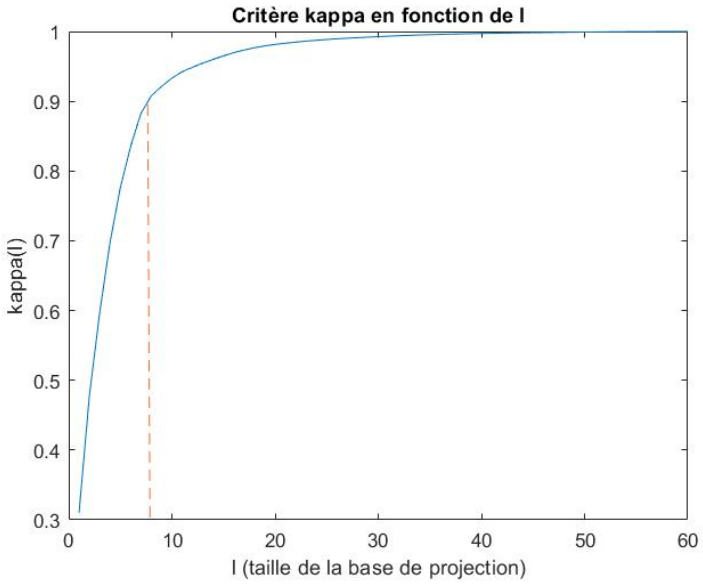

FIGURE 5 – Evolution du critère kappa en fonction de l

Pour la base d'apprentissage 1, on obtient pour taille optimale de la base de projection $l^* = 8$. Lorsque qu'on regarde la reconstruction d'une image pour cette valeur de l, on observe qu'elle est proche de la réalité. Cela permet donc une bonne qualité de reconstruction tout en optimisant les performances du programme.

2.4 Limitation du modèle

Partie non traitée par manque de temps

3 Classification

Afin de réaliser la reconnaissance de visages, c'est à dire associer un visage inconnu à un visage de la base connu le plus ressemblant, nous allons implémenter un classifieur. Un classifieur prend en entrée un vecteur représentant une image d'un visage et renvoie en sortie une classe d'image comprise dans la base de données d'entrainement.

3.1 Classifieur k-NN

Dans cette partie, nous allons implémenter un classifieur k_{NN} : k plus proches voisins. Ce classifieur compare les l composantes principales de l'image en entrée aux composantes principales de chaque image contenue dans la base

d'entrainement. Mathématiquement, les k plus proches voisins correspondent aux images (vecteurs) minimisant ce critère :

$$||w(x) - w(x_i)||_2 \tag{7}$$

où x est le vecteur en entrée et x_i (i est compris entre 1 et n) le vecteur i de la base de données d'entrainement.

3.1.1 Présentation du classifieur k-NN

L'implémentation du classifieur k_{NN} est réalisée dans le fichier *knn_ classifier.m*. Pour une image en entrée, on calcule ses composantes principales $w(x_i)$. Ensuite, on compare les composantes principales de l'image avec celles de chaque image comprise dans la base d'apprentissage. Les **k composantes principales** les plus proches de celle de l'image en entrée sont conservées. Finalement, nous déterminons la classe de l'image en fonction des classes des k plus proches voisins. La classe la plus présente étant attribuée à l'image d'entrée.

Cette version du classifieur k_{NN} pose le problème du choix de l'hyperparamètre k. En effet, il nous faut choisir le nombre de voisin avec lequel nous voulons procéder. Dans la suite de cette partie, nous choisirons k=5.

3.1.2 Calcul des matrices de confusion et du taux d'erreur global

Pour mesurer les performances du classifieur k_{NN}, nous utilisons la matrice de confusion qui permet d'illustrer le taux de réussite dans la détermination de la classe d'une image. Une performance maximale correspondant à la matrice identité. De cette approche, nous pouvons aussi déterminer le taux d'erreur. Ces deux métriques de performance du classifieur seront utilisées sur les 6 bases de test fournies. Les bases de test étant complexifiées graduellement, on s'attend à ce que le taux d'erreur augmente en fonction du numéro de la base.

FIGURE 6 – Base de test 3 et 5

Pour **les deux premières** bases de tests, nous obtenons la matrice identité et donc un taux d'erreur de 0. Voici les résultats pour les bases de 3 à 5 :

```
Conf_Mat =

    0.8182         0    0.0909         0         0    0.0909
         0    0.9091         0    0.0909         0         0
         0         0    1.0000         0         0         0
         0         0         0    1.0000         0         0
         0         0    0.0909    0.0909    0.8182         0
         0         0    0.1818         0         0    0.8182

Err_Rate =

    0.1061
```

```
Conf_Mat =

    0.3333         0    0.5556    0.1111         0         0
         0    0.6667    0.1111    0.2222         0         0
         0         0    0.8889    0.1111         0         0
         0         0         0    1.0000         0         0
         0         0    0.5556    0.1111    0.2222    0.1111
         0         0    0.3333    0.4444         0    0.2222

Err_Rate =

    0.4444
```

FIGURE 7 – Performances du classifieur k-NN sur les bases de test 3 et 4

```
Conf_Mat =

         0         0    0.8667    0.1333         0         0
         0    0.0667    0.3333    0.6000         0         0
         0         0    0.8667    0.1333         0         0
         0         0    0.0667    0.9333         0         0
         0         0    0.8000    0.2000         0         0
         0         0    0.0667    0.8667         0    0.0667

Err_Rate =

    0.6778
```

```
Conf_Mat =

         0         0         0    0.9167         0    0.0833
         0         0    0.1667    0.8333         0         0
         0         0    0.4167    0.5833         0         0
         0         0    0.1667    0.8333         0         0
         0         0    0.5833    0.4167         0         0
         0         0         0    1.0000         0         0

Err_Rate =

    0.7917
```

FIGURE 8 – Performances du classifieur k-NN sur les bases de test 5 et 6

On peut constater une baisse de performance proportionnelle à la détérioration des images dans les bases de test. Ce résultat est cohérent puisque la détérioration d'une image la rend plus difficile à associer à une image comprise dans la base d'apprentissage.

3.2 Classifieur gaussien

3.2.1 Présentation du classifieur gaussien

Afin de remédier aux problèmes que pose le classifieur k_{NN} classique, il serait intéressant d'introduire un modèle statistique sur les composantes principales. Pour cela, nous supposerons que les composantes principales d'une image de la classe i suivront une loi gaussienne multivariée de moyenne μ_i et de matrice de covariance Σ. La détermination de la classe d'une image en entrée se fera alors par la méthode du maximum de vraissemblance :

$$\phi(x) = \underset{j\in[1,m]}{argmax} f_j(w(x)) \tag{8}$$

où f_j est la densité de la loi gaussienne multivariée.

La loi gausienne multivariée dépendra notamment d'une moyenne μ_i correspondant à la classe d'image i et d'une matrice de covariance Σ. Ces deux paramètre seront déterminés à l'aide des données comprises dans la base d'apprentissage.

3.2.2 Représentation des nuages de points des couples de composantes principales (1,2),(2,3),(3,4),(4,5) pour toutes les images pour 3 individus sélectionnés

Dans un premier temps, nous affichons les nuages de points correspondants aux couples de composantes principales pour 3 individus de la base d'apprentissage. Nous avons choisi de réaliser cette tâche sur les trois premiers individus de la base d'appretissage. Un individu étant représenté par une couleur.

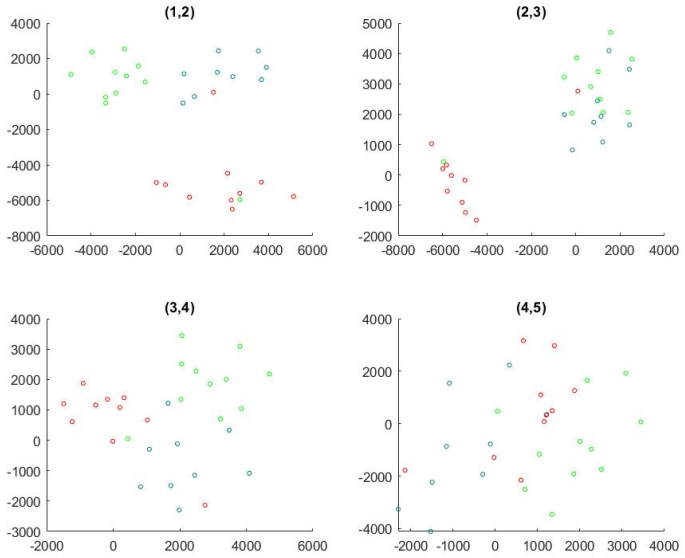

FIGURE 9 – Représentation des nuages de points des couples de composantes principales (1,2),(2,3),(3,4),(4,5) pour toutes les images pour 3 individus sélectionnés

Les nuages de points obtenus montrent que la répartition des couples de composantes principales forme des regroupements pour chaque individu. Cela démontre que les caractéristiques statistiques de chaque individu sont bien différenciables afin de les exploiter.

3.2.3 Calcul des matrices de confusion et du taux d'erreur global

De la même manière que le classifieur classique présenté dans la Partie 3.1, nous utilisons les matrices de confusion et le taux d'erreur pour observer les performances du classifieur gaussien. Pour chaque base de test disponible, une matrice de confusion et un taux d'erreur ont été calculé.

Pour les deux premières bases de test, tout comme le classifieur classique, le classifieur gaussien a un taux d'erreur de 0 et la matrice de confusion identité. Voici les résultats obtenus pour les 4 autres bases de test.

```
Conf_Mat_gauss =

    1.0000         0         0         0         0         0
         0    0.9091    0.0909         0         0         0
         0         0    1.0000         0         0         0
         0         0         0    1.0000         0         0
         0         0         0         0    1.0000         0
         0         0    0.0909         0         0    0.9091

Err_Rate_gauss =

    0.0303
```

```
Conf_Mat_gauss =

    0.8889         0    0.1111         0         0         0
         0    0.6667    0.3333         0         0         0
         0         0    1.0000         0         0         0
         0         0    0.2222    0.7778         0         0
         0         0    0.2222         0    0.7778         0
    0.1111         0    0.3333         0         0    0.5556

Err_Rate_gauss =

    0.2222
```

FIGURE 10 – Performances du classifieur k-NN gaussien sur les bases de test 3 et 4

```
Conf_Mat_gauss =

    0.0667         0    0.8667         0         0    0.0667
         0    0.0667    0.9333         0         0         0
         0         0    1.0000         0         0         0
         0         0    1.0000         0         0         0
         0         0    1.0000         0         0         0
    0.0667         0    0.9333         0         0         0

Err_Rate_gauss =

    0.8111
```

```
Conf_Mat_gauss =

         0         0    0.9167         0         0    0.0833
         0         0    1.0000         0         0         0
         0         0    1.0000         0         0         0
         0         0    1.0000         0         0         0
         0         0    1.0000         0         0         0
         0         0    1.0000         0         0         0

Err_Rate_gauss =

    0.8333
```

FIGURE 11 – Performances du classifieur k-NN gaussien sur les bases de test 5 et 6

De la même manière que le classifieur présenté Partie 3.1, les performances du classifieur gaussien diminue proportionnellement à la déterioration des images de la base de test.

3.3 Comparaison des performances entre les deux classifieurs

Afin de comparer les performances des deux classifieurs, nous avons tracé l'évolution du taux d'erreur en fonction du numéro de la base de test. Ainsi, nous pouvons comparer les performances des deux classifieurs pour chaque base de test.

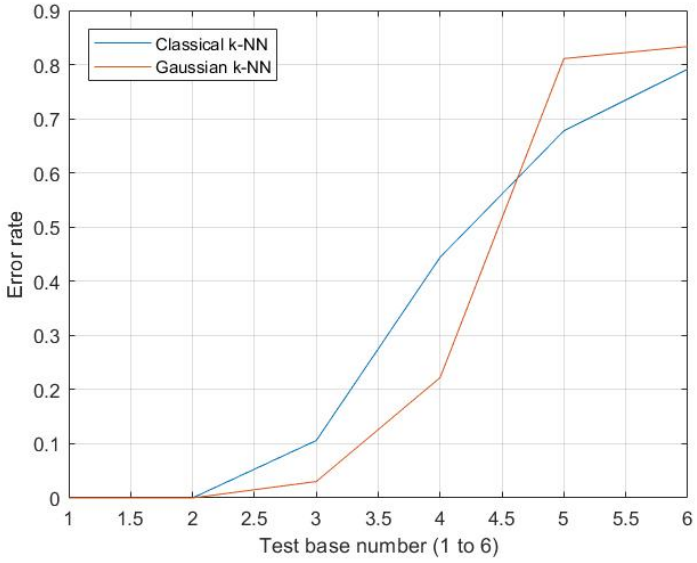

FIGURE 12 – Evolution du taux d'erreur en fonction du numéro de la base de test

On peut observer que le classifieur gaussien est plus performant pour les bases de tests 1 à 4. Cependant, il perd en performance dès la base de test numéro 5 et devient moins performant que le classifieur classique. Les bases de test 5 et 6 étant très dégradées, ce n'est pas abhérant d'avoir, pour les deux classifieurs, des résultats peu performants.

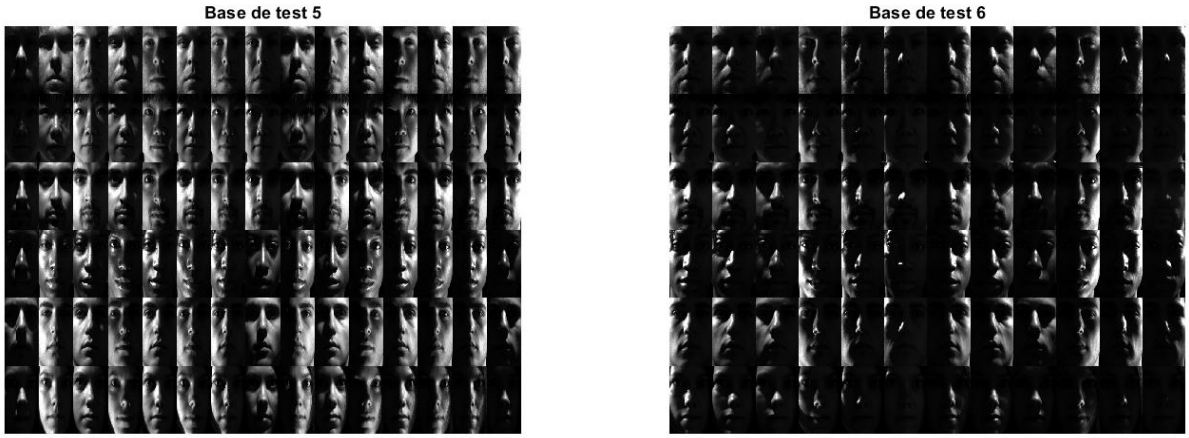

FIGURE 13 – Bases de test 5 et 6 fortement dégradées

3.4 Bonus

Afin de détecter si un visage n'appartient pas à la base de donnée, il est possible d'intégrer un test d'appartenance à partir d'un seuil de ressemblance (on peut parler de taux de confiance). Si le visage testé possède un pourcentage de ressemblance au dessus de ce seuil, alors le visage appartient à la base de données et est associé à l'un des visages, sinon le visage est un visage inconnu.

Malheureusement, nous n'avons pas pu implémenter cette fonctionnalité par manque de temps.

Conclusion

Lors de ce TP, nous avons réussi à implémenter des algorithmes en s'appuyant sur l'ACP pour reconstruire des visages. Nous avons constaté que plus la dimension du facespace l augmente, meilleure est la reconstruction de l'image. Cette meilleure qualité de reproduction d'une image diminue néanmoins les performances du programme car la taille de la base de projection est plus grande. Par conséquent, nous avons cherché à optimiser les performances du programme tout en garantissant une bonne qualité de reconstruction en faisant varier l. Pour la base d'apprentissage 1, la valeur optimale est $l^* = 8$.
Enfin, nous nous sommes intéressés à la classification des visages et à leurs performances. Pour cela, nous avons travaillé sur un classifieur k-NN et un classifieur gaussien. Le classifieur gaussien permettant d'attribuer un modèle statistique aux données et d'enlever le choix de l'hyperparamètre **k**. La comparaison des performances des deux classifieurs a montré une détection légèrement meilleure pour le classifieur gaussien.