المعهد الوطني للبريد والمواصلات
ⵓⵔⵣⵣⵓ ⵓⵏⵎⵙⵙⵓ ⵏ ⵜⵓⵍⵓⵙⵜ ⴰ ⵙⵎⵙⵙⵓⵍⵓⵖ
Institut National des Postes et Télécommunications

Rapport du projet

CONCEPTION DES ALGORITHMES LINEAIRES ET NON-LINEAIRE POUR LA RECONNAISSANCE FACIALE

Réalisé par

Jamal Ait Salh

Younes Bougrine

Encadré par

Mr. Mohammed Fekri

ASEDS INE1 2020/2021

Table des matières

1. Introduction générale 2
2. L'Analyse Discriminante Linéaire (ADL) 3
2.1. Reconnaissance par Fisherfaces 3
3. Analyse en Composantes Principales 6
3.1. Reconnaissance par Eigenfaces 6
4. L'analyse discriminante non linéaire : (Kernel FDA) 10
4.1. La méthode à noyau (Kernel) : 10
4.2. Kernel Fisher Discriminant Analysis : 11
4.2.1. Dispersion de deux classes dans un espace unidimensionnel 12
4.2.2. Cas de dispersion de multi-classes dans un espace multidimensionnel 14
5. Expérimentation et résultat 16
5.1. Introduction 16
5.2. Environnement de travail 16
5.2.1. Bases de données 16
5.2.2. Matériel utilisé 16
5.2.3. Environnement logiciel 17
5.2.4. Langages de programmation 17
5.3. Phase d'apprentissage 17
6. Conclusion générale 23
Référence 24

1.Introduction générale

De nos jours on parle de plus en plus de l'insécurité dans divers secteurs ainsi que des moyens informatiques à mettre en œuvre pour contrer cette tendance. La vérification et l'identification des individus est l'un des moyens permettant d'assurer cette sécurité. L'être humain se sert quotidiennement de son système visuel pour identifier les personnes de façon automatique, bien que le processus mis en jeu soit complexe.

L'homme a mis en place des moyens de vérification d'identité qui sont liés, soit à ce que possède une personne telle qu'une carte d'identité ou un passeport, soit à ce que sait cette personne, c'est le cas du mot de passe ou un code PIN. Néanmoins, ces éléments peuvent être oubliés, volés ou falsifiés. Pour contourner ces limitations, un autre moyen de sécurité a été développé qui permet d'utiliser, non pas l'information qu'un individu possède ou connaît, mais une information (propre) intrinsèque à cette personne. Cette nouvelle façon d'identification des individus est la biométrie visuelle.

Nous intéressons dans ce rapport à l'étude des caractéristiques des visages humaines, donc on va implémenter des algorithmes capables de reconnaitre le visage des personnes, Nous commençons d'abord par une étude théorique de quelques algorithmes de reconnaissance faciale. Puis une implémentation de ces algorithmes pour évaluer le bon fonctionnement de celles-ci.

Nous espérons que vous prenez autant de plaisir à lire ce rapport que nous avons pris tout au long du déroulement de ce projet.

2.L'Analyse Discriminante Linéaire (ADL)

L'Analyse Discriminante Linéaire a été introduite par Robert Fisher en 1936. Il s'agit d'expliquer et de prédire l'appartenance d'un individu à une classe (groupe) prédéfinie à partir de ses caractéristiques mesurées à l'aide de variables prédictives, l'ADL construit un sous-espace discriminant pour distinguer de façon optimale les visages de différentes personnes. Elle permet donc d'effectuer une véritable séparation de classes. Pour cela, elle optimise les variations inter-personnes par rapport aux variations intra-personnes. Donc, pour pouvoir l'utiliser, il faut au préalable organiser la base d'apprentissage d'images en plusieurs classes : une classe par personne et plusieurs images par classe.

2.1. Reconnaissance par Fisherfaces

L'algorithme de reconnaissance de visage *Fisherfaces* est né des travaux de *Belhumeur* de l'Université Yale (New Haven, USA), en 1997. Il utilise la technique d'Analyse Discriminante Linéaire *(Linear Discriminent Analysis* LDA).

L'algorithme LDA effectue une véritable séparation de classes. Donc, pour pouvoir l'utiliser, il faut organiser la base d'images d'apprentissage en plusieurs classes : une classe par personne et plusieurs images par classe. Le LDA analyse les vecteurs propres de la matrice de dispersion des données, avec pour objectif de maximiser les variations inter-classes, tout en minimisant les variations intra-classes.

Les étapes de l'algorithme LDA se décrivent comme suit :

i. Une image Ii(m,n) est traitée comme un vecteur Γi (m×n,1) dans un espace vectoriel de grande dimension (N = m × n), par concaténation des colonnes.

$$I_1 = \begin{pmatrix} a_{1,1} & \cdots & a_{1,m} \\ \vdots & \ddots & \vdots \\ a_{n,1} & \cdots & a_{n,m} \end{pmatrix} \Longrightarrow \Gamma_1 = \begin{pmatrix} a_{1,1} \\ \vdots \\ a_{n,1} \\ \vdots \\ a_{1,m} \\ \vdots \\ a_{n,m} \end{pmatrix}$$

ii. Après avoir rassemblé les M images dans une unique matrice, nous obtenons une

matrice d'images Γ, où chaque colonne représente une image Γi .

$$\Gamma = \begin{pmatrix} a_{1,1} & b_{1,1} & \cdots & z_{1,1} \\ \vdots & \vdots & \cdots & \vdots \\ a_{n,1} & b_{n,1} & \cdots & z_{n,1} \\ \vdots & \vdots & \cdots & \vdots \\ a_{1,m} & b_{1,m} & \cdots & z_{1,m} \\ \vdots & \vdots & \cdots & \vdots \\ a_{n,m} & b_{n,m} & \cdots & z_{n,m} \end{pmatrix}$$

iii. Nous calculons ensuite l'image moyenne Ψ de toutes les images collectées. Cette image peut être vue comme le centre de gravité du jeu d'images :

$$\Psi = \frac{1}{M}\sum_{i=1}^{M}\Gamma_i$$

iv. Ensuite, pour chaque classe C_i , nous calculons l'image moyenne Ψ_{C_i} :

$$\Psi_{C_i} = \frac{1}{q_i}\sum_{k=1}^{q_i}\Gamma_k$$

Avec qi , le nombre d'images dans la classe Ci .

v. Chaque image Γi de chaque classe C_i est ensuite recentrée par rapport à la moyenne. Nous obtenons, alors, une nouvelle image Φi telle que :

$$\Phi_i = \Gamma_i - \Psi_{C_i}$$

vi. Vient ensuite le calcul de nos différentes matrices de dispersion. Nous noterons : c le nombre total de classes (ïe. Le nombre d'individus dans la base d'apprentissage), q_i le nombre d'images dans la classe C_i et M le nombre total d'images.

- La matrice de Dispersion Intra-Classe ("within-class scatter matrix") (Sw) est donnée par :

$$S_w = \sum_{i=1}^{c}\sum_{\Gamma_k \in C_i}(\Gamma_k - \Psi_{C_i})(\Gamma_k - \Psi_{C_i})^T$$

- Et celle de la matrice de Dispersion Inter-Classe ("between-class scatter matrix") (Sb) est :

$$S_b = \sum_{i=1}^{c} q_i(\Psi_{C_i} - \Psi)(\Psi_{C_i} - \Psi)^T$$

- La matrice de Dispersion Totale (ST) :

$$S_T = \sum_{i=1} (\Gamma_i - \Psi)(\Gamma_i - \Psi)^T$$

vii. Une fois ces matrices calculées, nous devons trouver une projection optimale W qui minimise la dispersion intra-classe, relative à la matrice Sw, tout en maximisant la dispersion inter-classe, relative à la matrice Sb.

En d'autres termes, nous devons retrouver la matrice W qui maximise le critère d'optimisation de Fisher :

$$J(W) = \frac{W^T.S_b.W}{W^T.S_w.W}$$

Donc, W est optimale pour :

$$W_{opt} = arg \max_{W^T} \frac{|W^T.S_b.W|}{|W^T.S_w.W|}$$

W peut alors être trouvé en résolvant le problème généralisé aux valeurs propres :

$$S_w^{-1} S_b w = \lambda w$$

Donc W est égale aux vecteurs propres de la matrice : $S_W^{-1} S_B$

Pour maximiser J(W), il faut que la matrice Sw soit inversible, et ce qui n'est pas toujours le cas pour les applications de reconnaissance de visage, car le nombre d'images est généralement inférieur à la dimension des images dans la base de données. Ce problème est appelé « Small Sample Size ». Pour contourner ce problème, plusieurs approches ont été proposées mais la plus utilisée étant de réaliser une Analyse en Composante Principale au préalable pour diminuer la dimension des échantillons.

Ainsi, la projection vectorielle d'une image apprise réajustée par rapport à la moyenne Φi est définie par :

$$g(\Phi_i) = W^T \Phi_i$$

La phase de reconnaissance d'une image test Φt s'effectue en projetant Φt sur W^T :

$$g(\Phi_t) = W^T \Phi_t$$

Enfin, on effectue une mesure de distance entre l'image test et l'image projetée sur l'espace vectoriel engendré par W^T. Dans notre cas on a utilisé la distance Euclidienne.

Pour résumer cette méthode, étant donné une base de données contenant plusieurs images de plusieurs personnes, voici une récapitulation des principales étapes de l'algorithme de «Fisherfaces» :

1. Rassemblement de toutes les images de la base de données pour former une matrice.
2. Calcul de l'image moyenne totale et l'image moyenne de chaque classe.
3. Calcul des matrices de dispersion Sw et Sb.
4. Optimisation de critère de Fisher, pour trouver le nouvel espace de projection.
5. Projection de toutes les images de la base de données sur la nouvelle espace de projection.
6. Pour effectuer la reconnaissance, le visage test est projeté sur le nouvel espace de projection et nous calculons sa distance par rapport à toutes les images projetées sur ce nouvel espace.

3.Analyse en Composantes Principales

L'algorithme PCA est né des travaux de MA. Turk et AP. Pentland au MIT Media Lab, en 1991.

L'algorithme ACP, PCA en anglais (Principal Component Analysis) et aussi connu sous le nom d'Eigenfaces puisqu'il utilise des vecteurs propres et des valeurs propres. (Respectivement Eigenvectors et Eigenvalues en anglais).

Cet algorithme s'appuie sur des propriétés statistiques bien connues et utilise l'algèbre linéaire. Il est relativement rapide à mettre en œuvre, Il est à la base de nombreux algorithmes globaux actuels. L'idée principale consiste à exprimer les M images d'apprentissage selon une base de vecteurs orthogonaux particuliers, contenant des informations indépendantes d'un vecteur à l'autre. Ces nouvelles données sont donc exprimées d'une manière plus appropriée à la reconnaissance du visage.

3.1. Reconnaissance par Eigenfaces

La méthode de reconnaissance faciale Eigenfaces utilise la technique de l'Analyse en Composante Principale (ACP). Cette méthode est qualifiée de globale, car tout l'ensemble du visage est analysé. Elle est le fruit des travaux de Turk et Pentland.

Plus formellement, voici les principales étapes de la méthode ACP :

i. Tout comme dans le LDA, nous rassemblons les images de la base d'apprentissage dans une grande matrice d'images Γ où chaque colonne représente une image Γi, puis nous calculons l'image moyenne Ψ.

ii. Nous ajustons ensuite les données par rapport à la moyenne. L'image moyenne est alors soustraite de chaque image avec la formule suivante :

$$\Phi_i = \Gamma_i - \Psi, i = 1, ..., M$$

iii. Nous calculons ensuite la matrice de covariance du jeu de données. Cette matrice peut être vue comme une matrice de moments d'ordre 2 :

$$C = \sum^{M} \Phi_i \Phi_i^T = AA^T, avecA = [\Phi_1 \Phi_2 ... \Phi_M]$$

iv. La prochaine étape consiste à calculer les valeurs propres de cette matrice de covariance C de taille (N × N), c'est-à-dire, de l'ordre de la résolution des images, et leurs vecteurs propres associés. Le problème est que cela peut parfois être très difficile et très long. En effet, pour les images de résolution 320×240, nous devrions trouver les valeurs propres d'une matrice d'ordre 76800×76800, qui rendra la tâche de détermination des N^2 vecteurs propres intraitables et infaisables en calcul.
C'est à présent que nous allons réduire l'information en limitant les composants avec lesquels nous travaillerons pour que nous puissions accélérer le calcul, en accord avec le principe de l'analyse en composantes principales. Voici comment donc procéder pour accélérer les calculs :

- Considérons une matrice E et λi un de ses valeurs propres tels que :

$$E = A^T.A$$

Cette matrice est donc d'ordre M × M avec M c'est le nombre des colonnes dans Γ.

- Considérons aussi le vecteur ei, vecteur propre de E associé à λi . Alors nous avons la relation :

$$E.e_i = \lambda_i.e_i.$$

- Puisque E = AT.A, nous avons donc :

$$E.e_i = A^T.A.e_i = \lambda_i.e_i.$$

- Multiplions cette égalité par A :

$$A.E.e_i = A.A^T.A.e^i = A.\lambda_i.e_i.$$

- Comme C = A.AT , nous avons donc :

$$A.E.e_i = C.A.e_i = \lambda_i.A.e_i.$$

Nous en déduisons donc qu'avec ei vecteur propre de la matrice E associé à la valeur propre λi, nous disons, par conséquent, que A.ei est un vecteur propre de la matrice C associée à la même valeur propre λi .

Nous pouvons donc trouver les valeurs propres de C en trouvant les valeurs propres d'une matrice E plus petite. Pour trouver les vecteurs propres de C, il suffit juste de multiplier les vecteurs propres de E par la matrice A.

v. Les vecteurs propres trouvés sont ensuite, ordonnés selon leurs valeurs propres correspondantes, de manière décroissante. Plus une valeur propre est grande, plus la variance capturée par le vecteur propre est importante

vi. Nous définissons alors un espace vectoriel engendré par ce M' (M' : le nombre des plus grandes valeurs propres de C) vecteurs propres, que l'on appelle l'espace de visage ("Face Space").

vii. Une fois que l'apprentissage est fait, on passe à la reconnaissance, on projette l'ensemble des images dans le nouvel espace généré par les vecteurs propres du C

La projection vectorielle d'une image apprise réajustée par rapport à la moyenne Ψ est définie par :

$$g(\Phi_i) = W^T \Phi_i$$

Avec W c'est la matrice des vecteurs propres de C

La phase de reconnaissance d'une image test Φt s'effectue en projetant Φt sur W^T :

$$g(\Phi_t) = W^T \Phi_t$$

Enfin, on effectue une mesure de distance entre l'image test et l'image projetée sur l'espace vectoriel engendré par W^T. Dans notre cas on a utilisé la distance Euclidienne.

4.L'analyse discriminante non linéaire : (Kernel FDA)

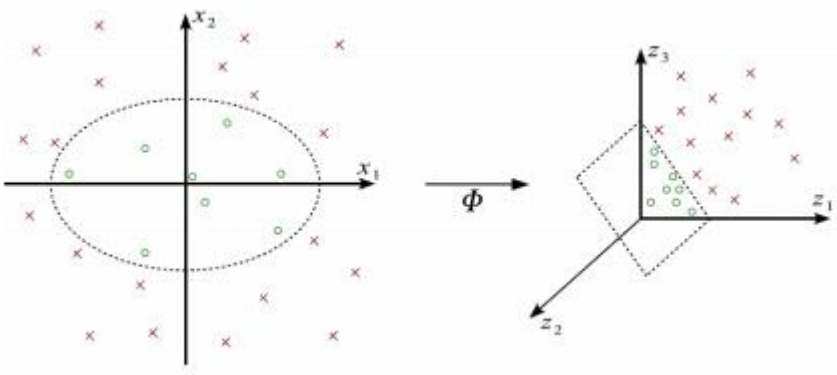
4.1. La méthode à noyau (Kernel) :

De manière générale, les méthodes de projections non linéaires servent à représenter des espaces de données plus complexes que les techniques linéaires du Fisher Discriminant Analysis présentées dans la section précédente. En effet, ces méthodes non linéaires cherchent l'espace de représentation $\mathcal{F}$ dans lequel les données seront projetées.

Parmi les techniques non linéaires les plus utilisées, la méthode du Kernel est la plus courante dans la reconnaissance des formes en général et la reconnaissance faciale en particulier.

Cette méthode Kernel sert à ramener le problème de classification dans un espace $\mathcal{F}$ de très grande dimension où l'on espère que les données complexes seront classées.

Afin de bien comprendre le problème que la méthode du Kernel essaie de résoudre, on prend le cas de deux classes qui ne sont pas linéairement séparables dans l'espace



initial des données. Si on choisit une fonction de transformation φ : $R^2 \rightarrow R^3$ les données projetées φ(x) peuvent être linéairement séparables dans R^3 (Figure 1) :

L'espace F peut être de taille très grande et il serait donc difficile de travailler directement sur cet espace. Alors que grâce à l'astuce du Kernel, il n'est pas nécessaire d'expliciter l'espace F, ni la fonction φ. On a seulement besoin de travailler avec un produit scalaire sur F noté k :

$$\begin{aligned} k: \quad \Re^d \times \Re^d &\rightarrow \Re \\ (x,y) &\rightarrow k(x,y) =< \phi(x), \phi(y) >= (\phi(x))^t \phi(y) \end{aligned}$$

Qui est une mesure de similarité entre les deux vecteurs parce que le produit intérieur capture la similitude. Le produit scalaire k est appelé fonction noyau. Notez qu'on peut calculer de même le kernel de deux matrices et obtenir une matrice kernel qui est applé autrement matrice de Gram, donné par : $X_1 \in F^{d \times n1}$ et $X_2 \in F^{d \times n2}$

$$\mathbb{R}^{n_1 \times n_2} \ni \boldsymbol{K}(\boldsymbol{X}_1, \boldsymbol{X}_2) := \boldsymbol{\Phi}(\boldsymbol{X}_1)^\top \boldsymbol{\Phi}(\boldsymbol{X}_2),$$

Où $\boldsymbol{\Phi}(\boldsymbol{X}_1) := [\phi(\boldsymbol{x}_1), \ldots, \phi(\boldsymbol{x}_n)]$ est la matrice des données projetés dans L'espace caractéristique F. De même pour $\Phi(X_2)$.

Il existe de nombreux noyaux, les plus utilisés pour la reconnaissance faciale sont :

$$\begin{aligned}
&\text{Linear: } k(\boldsymbol{x}_1, \boldsymbol{x}_2) = \boldsymbol{x}_1^\top \boldsymbol{x}_2 + c_1, \\
&\text{Polynomial: } k(\boldsymbol{x}_1, \boldsymbol{x}_2) = (c_1 \boldsymbol{x}_1^\top \boldsymbol{x}_2 + c_2)^{c_3}, \\
&\text{Gaussian: } k(\boldsymbol{x}_1, \boldsymbol{x}_2) = \exp\left(-\frac{||\boldsymbol{x}_1 - \boldsymbol{x}_2||_2^2}{2\sigma^2}\right), \\
&\text{Sigmoid: } k(\boldsymbol{x}_1, \boldsymbol{x}_2) = \tanh(c_1 \boldsymbol{x}_1^\top \boldsymbol{x}_2 + c_2),
\end{aligned}$$

Où c1, c2, c3 et sont des constantes scalaires et II.II est la norme Euclidienne. La Gaussienne et les noyaux sigmoïdes sont respectivement appelés fonction de base radiale (RBF) et tangente hyperbolique.

4.2. Kernel Fisher Discriminant Analysis :

Clairement pour la plupart des données du monde réel, un discriminant linéaire n'est pas assez complexe, pour augmenter l'expressivité du discriminant, le kernel FDA nous aide à trouver des directions non linéaires en projetant d'abord les données dans un espace de caractéristiques de grande dimension F en utilisant une transformation non linéaire φ.

Une fois cette projection est faite, les méthodes linéaires classiques du FDA seront utilisées avec leur critère d'optimisation dans F afin de trouver les caractéristiques non linéaires optimales.

4.2.1. Dispersion de deux classes dans un espace unidimensionnel

On va étudier dans un premier temps le kernel FDA pour deux classes dans un espace de Hilbert de dimension 1, ensuite on va conclure pour le cas le plus généralisé de plusieurs classes dans un espace multidimensionnel.

Données d'initialisation :

Soit φ une application non linéaire qui projette l'ensemble des données X dans l'espace caractéristique **H** (espace de Hilbert).

Soient **t** la dimension de l'espace H et $\mathbf{n_j}$ le nombre d'éléments de la classe j.

Soit c le nombre de classe étudié.

Recherche de solution :

La distance d_B dans le nouvel espace H devient :

$$d_B := \boldsymbol{\phi}(\boldsymbol{u})^\top \boldsymbol{\Phi}(\boldsymbol{S}_B)\, \boldsymbol{\phi}(\boldsymbol{u})$$

Et d'après la formule de S_B dans la phase précédente, on aura :

$$\boldsymbol{\Phi}(\boldsymbol{S}_B) := \\ (\boldsymbol{\phi}(\boldsymbol{\mu}_1) - \boldsymbol{\phi}(\boldsymbol{\mu}_2))\,(\boldsymbol{\phi}(\boldsymbol{\mu}_1) - \boldsymbol{\phi}(\boldsymbol{\mu}_2))^\top$$

Et on sait que toutes les solutions $\boldsymbol{\phi(\mu)}$ doit se situer dans la base des vecteurs de formation projetés dans l'espace caractéristique, i.e., $\boldsymbol{\Phi}(\boldsymbol{X}_1) := [\boldsymbol{\phi}(\boldsymbol{x}_1), \ldots, \boldsymbol{\phi}(\boldsymbol{x}_n)]$

Donc : $$\boldsymbol{\phi}(\boldsymbol{u}) = \sum^{n} \theta_i\, \boldsymbol{\phi}(\boldsymbol{x}_i) = \boldsymbol{\Phi}(\boldsymbol{X})\, \boldsymbol{\theta}$$

Avec $\mathbb{R}^n \ni \boldsymbol{\theta} := [\theta_1, \ldots, \theta_n]^\top$ est le vecteur inconnu de coefficients, et $\boldsymbol{\phi(\mu)}$ est la direction de Fisher tirée à l'espace caractéristique.

Enfin on peut trouver après certains calcul mathématique la nouvelle formule de d_B :

$$\boxed{d_B \overset{(99)}{=} \boldsymbol{\theta}^\top (\boldsymbol{m}_1 - \boldsymbol{m}_2)(\boldsymbol{m}_1 - \boldsymbol{m}_2)^\top \boldsymbol{\theta} = \boldsymbol{\theta}^\top \boldsymbol{M} \boldsymbol{\theta},}$$

Où $\mathbb{R}^{n\times n} \ni \boldsymbol{M} := (\boldsymbol{m}_1 - \boldsymbol{m}_2)(\boldsymbol{m}_1 - \boldsymbol{m}_2)^\top$ est la dispersion intérieure en kernel FDA

Et $\boldsymbol{m}_j(i) := \frac{1}{n_j}\sum_{k=1}^{n_j} k(\boldsymbol{x}_i, \boldsymbol{x}_k^{(j)}).$ appartenant à R^n.

De même, La distance d_w dans le nouvel espace H devient :

$$\mathbb{R} \ni d_W := \boldsymbol{\phi}(\boldsymbol{u})^\top \boldsymbol{\Phi}(\boldsymbol{S}_W)\, \boldsymbol{\phi}(\boldsymbol{u})$$

D'après la formule de S_w de ce qui précède, on a :

$$\boldsymbol{\Phi}(\boldsymbol{S}_W) := \sum_{j=1}^{c}\sum_{i=1}^{n_j} \left(\boldsymbol{\phi}(\boldsymbol{x}_i^{(j)}) - \boldsymbol{\phi}(\boldsymbol{\mu}_j)\right)\left(\boldsymbol{\phi}(\boldsymbol{x}_i^{(j)}) - \boldsymbol{\phi}(\boldsymbol{\mu}_j)\right)^\top.$$

Alors, en effectuant quelque calcul sur la distance d_W, on peut trouver enfin que :

$$\boxed{d_W = \boldsymbol{\phi}(\boldsymbol{u})^\top \boldsymbol{\Phi}(\boldsymbol{S}_W)\, \boldsymbol{\phi}(\boldsymbol{u}) = \boldsymbol{\theta}^\top \boldsymbol{N}\boldsymbol{\theta}.}$$

Avec $\mathbb{R}^{n\times n} \ni \boldsymbol{N} := \sum_{j=1}^{c} \boldsymbol{K}_j \boldsymbol{H}_j \boldsymbol{K}_j^\top,$ est la dispersion extérieure en kernel FDA.

Où $\mathbb{R}^{n_j\times n_j} \ni \boldsymbol{H}_j := \boldsymbol{I} - \frac{1}{n_j}\mathbf{1}\mathbf{1}^\top,$ et K_j est la matrice kernel de l'ensemble des données d'entraînement et les données d'entraînement de la classe j. L'élément (a,b) de la matrice K_j est donnée par :

$$\boldsymbol{K}_j(a,b) := k(\boldsymbol{x}_a, \boldsymbol{x}_b^{(j)}).$$

La matrice H_j est appelé la matrice de centrage.

Pour trouver le discriminant linéaire dans H, on doit, comme on a fait dans la phase précédente, maximiser le critère kernel de Fisher :

$$\boxed{f(\boldsymbol{\theta}) := \frac{d_B(\boldsymbol{\theta})}{d_W(\boldsymbol{\theta})} = \frac{\boldsymbol{\phi}(\boldsymbol{u})^\top \boldsymbol{\Phi}(\boldsymbol{S}_B)\, \boldsymbol{\phi}(\boldsymbol{u})}{\boldsymbol{\phi}(\boldsymbol{u})^\top \boldsymbol{\Phi}(\boldsymbol{S}_W)\, \boldsymbol{\phi}(\boldsymbol{u})} = \frac{\boldsymbol{\theta}^\top \boldsymbol{M}\boldsymbol{\theta}}{\boldsymbol{\theta}^\top \boldsymbol{N}\boldsymbol{\theta}},}$$

Avec $\Theta \in R^n$ est la direction du kernel Fisher.

Par analogie avec la solution de la première phase du discriminant linéaire du Fisher, la solution de maximisation du critère kernel Fisher est de trouver les vecteurs propres de la matrice $N^{-1}M$:

$$\boldsymbol{M\theta} = \lambda \boldsymbol{N\theta},$$

$$\boxed{\boldsymbol{\theta} = \mathbf{eig}(\boldsymbol{N}^{-1}\boldsymbol{M})}$$

qui est un problème aux valeurs propres généralisé (M,N). $\boldsymbol{\theta}$ est le vecteur propre avec la plus grande valeur propre (car l'optimisation est maximisation) et λ est la valeur propre correspondante.

$\boldsymbol{\theta}$ est la direction de kernel Fisher ou l'axe de kernel Fisher.

Similaire à FDA, une solution possible à la valeur propre généralisée problème (M, N) est :

Où eig(.) désigne le vecteur propre de la matrice avec la plus grande valeur propre.

Finalement, la projection d'un point x_i de données d'entrainement est donnée par :

$$\begin{aligned}\mathbb{R} \ni \phi(\tilde{\boldsymbol{x}}_i) &= \boldsymbol{\phi}(\boldsymbol{u})^\top \boldsymbol{\phi}(\boldsymbol{x}_i) \overset{(95)}{=} \boldsymbol{\theta}^\top \boldsymbol{\Phi}(\boldsymbol{X})^\top \boldsymbol{\phi}(\boldsymbol{x}_i) \\ &= \boldsymbol{\theta}^\top \boldsymbol{k}(\boldsymbol{X}, \boldsymbol{x}_i),\end{aligned}$$

Et pour la projection de l'ensemble des données d'entrainement :

$$\mathbb{R}^{1\times n} \ni \boldsymbol{\Phi}(\widetilde{\boldsymbol{X}}) = \boldsymbol{\theta}^\top \boldsymbol{K}(\boldsymbol{X}, \boldsymbol{X}),$$

4.2.2. Cas de dispersion de multi-classes dans un espace multidimensionnel

Soit p la dimension de l'espace caractéristique et n le nombre d'éléments des données d'entrainement.

Dans ce cas, la dispersion extérieure reste inchangée et garde la formule précédente. Tandis que la dispersion intérieure M est :

$$\mathbb{R}^{n\times n} \ni \boldsymbol{M} := \sum_{j=1}^{c} (\boldsymbol{m}_j - \boldsymbol{m}_*)(\boldsymbol{m}_j - \boldsymbol{m}_*)^\top$$

Avec $\boldsymbol{m}_*(i) := \frac{1}{n}\sum_{k=1}^{n} k(\boldsymbol{x}_i, \boldsymbol{x}_k).$

Dans le cas d'un espace kernel Fisher multidimensionnel, le critère KF devient :

$$f(\boldsymbol{\Theta}) := \frac{d_B(\boldsymbol{\Theta})}{d_W(\boldsymbol{\Theta})} = \frac{\mathbf{tr}\big(\boldsymbol{\phi}(\boldsymbol{U})^\top \boldsymbol{\Phi}(\boldsymbol{S}_B)\,\boldsymbol{\phi}(\boldsymbol{U})\big)}{\mathbf{tr}\big(\boldsymbol{\phi}(\boldsymbol{U})^\top \boldsymbol{\Phi}(\boldsymbol{S}_W)\,\boldsymbol{\phi}(\boldsymbol{U})\big)}$$
$$= \frac{\mathbf{tr}(\boldsymbol{\Theta}^\top \boldsymbol{M}\boldsymbol{\Theta})}{\mathbf{tr}(\boldsymbol{\Theta}^\top \boldsymbol{N}\boldsymbol{\Theta})},$$

Où les colonnes de Θ sont les directions de kernel Fisher $\mathbb{R}^{n\times p} \ni \boldsymbol{\Theta} = [\boldsymbol{\theta}_1, \ldots, \boldsymbol{\theta}_p]$

Similaire à l'équation vue précédemment, la solution pour maximiser ce critère est de trouver les vecteurs propres de la matrice $N^{-1}M$:

$$\boxed{\boldsymbol{M}\boldsymbol{\Theta} = \boldsymbol{N}\boldsymbol{\Theta}\boldsymbol{\Lambda},}$$

Le problème aux valeurs propres généralisé (M, N). Les colonnes de Θ sont les vecteurs propres triées de la plus grande à la plus petite valeur propre (car l'optimisation est la maximisation) et la diagonale de Ą sont les valeurs propres correspondantes.

Et donc la projection d'un point x_i de données d'entrainement est donnée par :

$$\mathbb{R}^p \ni \boldsymbol{\phi}(\widetilde{\boldsymbol{x}}_i) = \boldsymbol{\Phi}(\boldsymbol{U})^\top \boldsymbol{\phi}(\boldsymbol{x}_i) \overset{(96)}{=} \boldsymbol{\Theta}^\top \boldsymbol{\Phi}(\boldsymbol{X})^\top \boldsymbol{\phi}(\boldsymbol{x}_i)$$
$$= \boldsymbol{\Theta}^\top \boldsymbol{k}(\boldsymbol{X}, \boldsymbol{x}_i), \quad (1$$

Et pour la projection de l'ensemble des données d'entrainement :

$$\boxed{\mathbb{R}^{p\times n} \ni \boldsymbol{\Phi}(\widetilde{\boldsymbol{X}}) = \boldsymbol{\Theta}^\top \boldsymbol{K}(\boldsymbol{X}, \boldsymbol{X}),}$$

En kernel Fisher, on n'a pas de reconstruction.

Le rang de la matrice ***N*** est le min(c, n) au maximum puisqu'elle est une matrice carrée de dimension n x n et son calcul se fait à travers c itération. Donc le rang de N^{-1} est aussi min(c, n). Et le rang de M est min(n, c-1) au maximum.

Il faut noter donc que la dimension de l'espace kernel Fisher est **c-1** au maximum.

Les vecteurs propres principaux **c - 1** sont considéré comme les directions de kernel Fisher et le reste des vecteurs propres sont invalides et ignorés.

5.Expérimentation et résultat

5.1. Introduction

Dans cette partie, nous expérimenterons les méthodes de reconnaissance de visage que nous avons décrit dans la partie précédente. D'abord, nous présenterons les environnements matériel et logiciel que nous allons utiliser pour notre travail, puis la réalisation des expériences, et enfin, les résultats de ces expérimentations.

5.2. Environnement de travail

5.2.1. Bases de données

Pour que nous puissions comparer les différents algorithmes de reconnaissance de visage, la communauté scientifique a construit des bases de données ou base d'images. Une base d'image regroupe plusieurs images de plusieurs personnes. Actuellement, plusieurs bases d'images sont disponibles. Mais nous avons opté pour la base Yale « The Yale Face Database ».

Cette base contient 165 images de 15 personnes, chaque personne a 11 images différentes

FIGURE : Quelques images de la base Yale Dataset.

5.2.2. Matériel utilisé

Afin de mener à bien ce projet, il a été mis à notre disposition un ordinateur HP probook de configuration suivante :

- **Processeur** : Dual Core Intel Core i5.
- **RAM** : 8.00 Go.
- **Mémoire du disque dur** : 250 Go.
- **Système d'exploitation** : Microsoft Windows 10 Pro.

5.2.3. Environnement logiciel

Jupyter Notebook : est une application Web open source qui vous permet de créer et de partager des documents contenant du code en direct, des équations, des visualisations et du texte narratif. Les utilisations incluent : le nettoyage et la transformation des données, la simulation numérique, la modélisation statistique, la visualisation des données, l'apprentissage automatique.

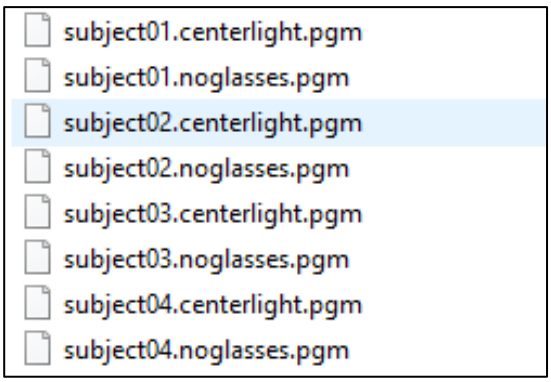
5.2.4. Langages de programmation

Python : est un langage de programmation interprété, multiparadigme et multiplateformes.

Il favorise la programmation impérative structurée, fonctionnelle et orientée objet

5.3. Phase d'apprentissage

Pour implémenter nos algorithmes, il nous faut d'abord concevoir une base de données contenant des images des personnes que nous souhaitons être reconnues par le système. Pour ce faire, nous devons créer manuellement les dossiers qui contient les images d'apprentissage et les images du test, un extrait du dossier test que nous avons fabriqué avec la base d'image Yale



Les chiffres que contient les noms des images « 0 », « 01 », «0 2 », . . ., sont des identifiants affectés à chaque personne de la base de données. Ils sont uniques pour les images d'une même personne.

Pour implémenter l'algorithme LDA, il faut a priori implémenter le PCA pour réduire les dimensions du jeu de données, Par suite on va procéder à l'implémentation des deux algorithme **PCA + LDA**

- **Prétraitement des données :**

On commence par importer tous les modules nécessaires pour le traitement et la lecture des données, notre Dataset contient deux dossiers qui à leur tour contient des sous dossiers pour les images d'apprentissages et un autre pour les images du test.

```
import os
import sys
import cv2
import numpy as np
import matplotlib
import matplotlib.pyplot as plt
%matplotlib inline

'''_____________________________USER SELECTED AREA_____________________________'''
dirNameTrain = './INPUT/TRAIN'
dirNameTest      = './INPUT/TEST'
```

On prépare les données du test et du traitement, on met l'ensembles des données sous forme d'une seule matrice dont chaque colonne est une image, on fait cette opération pour train et test.

```
# Read images
train_images = ReadImages(dirNameTrain)
print(train_images[0])
#plt.imshow(train_images[0], cmap= 'gray')

# Create train face matrix.
face_vector = CreateDataMatrix(train_images)

    # Read test images
test_images = ReadImages(dirNameTest)

# Create test face matrix.
test_face_vector = CreateDataMatrix(test_images)
```

- **Apprentissage :**

On calcul l'image moyenne de toutes les images du train

```
# Find average_face_vector, sum(all image vectors)/number(images).
average_face_vector = np.mean(face_vector, axis=1)
average_face_vector.shape = (len(average_face_vector), 1)

# Size of images
sz = train_images[0].shape
print(sz)
number_train_images= face_vector.shape[1]
print(number_train_images)
#print(np.array(train_images))
```

Nous ajustons par la suite les données par rapport à la moyen et on calcul la matrice de covariance

```
# Subtract average_face_vector from every image vector.
sub_face_vector = np.zeros(face_vector.shape, dtype=np.float32)
sub_face_vector = face_vector - average_face_vector

# Calculate covariance matrix of above matrix -> C = A*transpose(A)
covariance_matrix = np.dot(sub_face_vector.T, sub_face_vector)
print("covariance_matrix.shape =", covariance_matrix.shape)
print(covariance_matrix)
```

On calcule les vecteurs propres et les valeurs propres de la matrice de covariance.

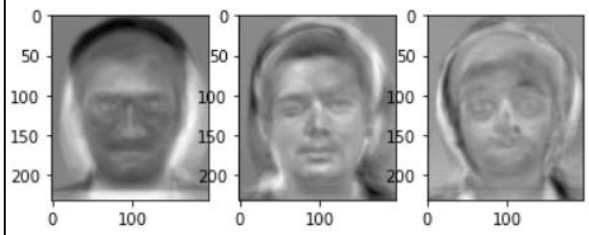
```
# Find eigenvectors and eigenvalues of above covariance matrix.
# eigenvalues arranged to match with associated eigenvector
print('Calculating PCA ', end=' ... ', flush=True)
eigenValues, eigenVectors = Find_Eigen(sub_face_vector)
print ('DONE')
print(eigenVectors.shape)

# Plot the ordered eigen values.
plt.plot(eigenValues, 'r+')
plt.title('eigenValues Ordered')
plt.ylabel('Value of eigenValues')
plt.xlabel('index number')
```

On utilise l'un des modules de python pour afficher l'image que contient l'un de ces vecteurs propres

```
plt.subplot(1,3,1)
plt.imshow(eigenVectors[:,0].reshape(sz), cmap= 'gray')
plt.subplot(1,3,2)
plt.imshow(eigenVectors[:,7].reshape(sz), cmap= 'gray')
plt.subplot(1,3,3)
plt.imshow(eigenVectors[:,10].reshape(sz), cmap= 'gray')
```

```
<matplotlib.image.AxesImage at 0x23f01374dc0>
```



On projette les données du train dans le nouvel espace engendré par la matrice transposé des vecteurs propres appelé « espace visage », et on calcule la matrice des poids qui correspondent à la projection des donnés dans l'espace visage.

```
# Calculate weights
weights = np.dot(eigenVectors.T, sub_face_vector)
weights.shape
```

Après l'application de l'algorithme PCA pour réduire les dimensions du jeu de données, on a obtenu une nouvelle matrice qui représentent notre jeu de donnés mais avec des dimensions plus réduites (i.e. la matrice des poids 'weights'), par la suite on va appliquer LDA sur la matrice 'weights'.

Premièrement, on calcul la moyenne de la matrice 'weights'.

```
# calculate mean of weights matrix
overall_mean = np.mean(weights, axis = 1)
overall_mean = overall_mean.reshape(overall_mean.shape[0],1)
print(overall_mean.shape)
```

Vient ensuite le calcul de nos différentes matrices de dispersion.

On dispose de 40 classes (i.e. 40 personnes) dans notre dataset, on commence par calculer la matrice de Dispersion Intra-Classe Sw, après la matrice de Dispersion Inter-Classe Sb.

```
#Finding the within class scatter matrix
#finding mean for individual classes
# number of classes are 40
SW = np.zeros([105,105])
for i in range(15):
    ind = i * 7
    V = weights[:,ind:ind+7]
    mean_local = np.mean(V, axis = 1)
    mean_local = mean_local.reshape(mean_local.shape[0],1)
    mean = np.repeat(mean_local, 7,axis = 1)
    diff = V - mean
    variance = np.dot(diff, diff.T)
    SW = SW + variance
```

```
#Finding the between class scatter matrix
SB = np.zeros([105,105])
print(variance.shape)
for i in range(15):
    j = i+7
    V = weights[:,i:j]
    mean_local = np.mean(V, axis = 1)
    mean_local = mean_local.reshape(mean_local.shape[0],1)
    diff = mean_local - overall_mean
    sigma = np.dot(diff, mean_local.T)
    SB = SB  + sigma
```

Maintenant c'est le temps de déterminer la solution optimale pour maximiser le critère de fisher, cette solution est déterminée à partir de la résolution du problème de vecteur propre de la matrice J

```
#finding the criterion function
#this function maximises the between class scatter and minimizes the within class scatter
J = np.dot(np.linalg.inv(SW), SB)
print(J.shape)
```

défini comme suit :

On calcul les valeurs propres et les vecteurs propres de la matrice J

```
#finding eigenvalues and eigenvectors of the matrix J
eigenval, eigenvec = np.linalg.eig(J)
```

Après le calcul de la matrice moyen pour chaque classe, et la matrice des poids recentré par rapport a cette matrice, on projette la matrice des poids recentré dans l'espace généré par la matrice solution de critère de fisher.

```
#calculate the mean of each classes
mean_classes = []
for i in range(15):
    ind = i * 7
    V = weights[:,ind:ind+7]
    mean_local = np.mean(V, axis = 1)
    mean_local = mean_local.reshape(mean_local.shape[0],1)
    mean_classes += [mean_local]*7
mean_classes = np.array(mean_classes).reshape(105,105)
print(mean_classes.shape)

# calculer la matrice des poids recentré par rapport à la moyen de chaque classe
weights_mean_for_each_class = weights - mean_classes

# on projete la matrice des poids recentré dans l'espace géneré par la matrice des vecteur propre de inv(Sw)*Sb
fisher_faces = np.dot(eigenvec.T, weights_mean_for_each_class)
print(fisher_faces.shape)
```

❖ Test :

On applique le PCA sur les données du test pour réduire leurs dimensions.

```
# Find average_face_vector, sum(all image vectors)/number(images).
average_test_face_vector = test_face_vector - average_face_vector

#Transform the test images to 'V' space
weights_test = np.dot(eigenVectors.T, average_test_face_vector)
print(weights_test.shape)
```

Après, l'obtention d'une matrice test dont les dimensions sont réduites, on applique alors la LDA sur cette matrice

```
# projeter weights_test dans l'espace géneré par eignvector(inv(Sw)*Sb)
projected_fisher_faces = np.dot(eigenvec.T, weights_test)
print(projected_fisher_faces.shape)
```

On utilise la distance euclidienne pour classifier les images du test.

```
    # Create array to keep matched images
    Matched_Faces = np.zeros((weights_test.shape[1]))

    # Compare all weights_test among the weights and Calculate Similarity
    for i in range(0, weights_test.shape[1]):
        error = np.zeros((weights.shape[1]))
        for j in range (0, weights.shape[1]):
            error[j] = np.sqrt(np.sum((weights[:, j] - weights_test[:, i])**2))

        # Match the test image with training image
        Matched_Faces[i] = error.argmin() // 7

    # Print the indexes of matched images
    print(Matched_Faces)
```

```
[ 0.  0.  1.  1.  2.  2.  3.  3.  4.  4.  5.  5. 13.  6.  7.  7.  8.  8.
  9.  9. 10. 10. 11. 11. 12. 12. 13. 13. 14. 14.]
```

Après la partie test, on a trouvé que notre modèle est capable de prédire l'identité de 29 personnes parmi 30 personnes.

Donc Accuracy = 97 %

6.Conclusion générale

Ce travail s'inscrit dans le domaine de la reconnaissance automatique des visages. Celui-ci consiste à vérifier l'identité d'une personne à partir de son image. Utilisés principalement pour des raisons de sécurité et/ou confidentialité, les systèmes de reconnaissance automatique des visages sont souvent développés dans les applications de télésurveillance et l'accès à des endroits sécurisés. On peut également l'associer à d'autre modalités dans des systèmes de vérification et d'authentification multimodale de l'identité.

Dans ce travail, il était question d'étudier les algorithmes de reconnaissance de visages basé sur les méthodes LDA et PCA. Les méthodes de projections peuvent être de nature linéaire ou non linéaire mais ayant un but commun. C'est la projection des données dans un nouveau sous-espace, qui élimine les éléments ne portant pas une information aidant à classifier les

visages Nous avons en extension, fusionné les méthodes avec le classificateur classique : la distance euclidienne, ce système atteint un taux de reconnaissance = 97%.

Référence

1. M. Donias, « Caractérisation de Champs d'Orientation par Analyse en Composantes Principales et Estimation de la Courbure : Application aux Images Sismiques », Thèse de doctorat, Université Bordeaux I, France, Janvier 1999.

2. Cherng Jye Liou, "A Real Time Face Recognition System", DSP/IC Design Lab, Department of Electrical Engineering, National Taiwan University, June 1997.

3. Golub G.H., "The Generalized Eigenvalue Problem", Lectures on Matrix Computation, Ph.D program of the Dipartimento di Matematica "Istituto Guido Castelnuovo", Lecture No. 11, Roma, 2004.

4. https://pythonmachinelearning.pro/face-recognition-with-eigenfaces/

5. https://iopscience.iop.org/article/10.1088/1742-6596/1196/1/012010/pdf

6. https://ieeexplore.ieee.org/document/7154774

7. https://www.researchgate.net/publication/275330326_FACE_RECOGNITION_USING_PCA_ALGORITHM
8. http://journal.uad.ac.id/index.php/TELKOMNIKA/article/view/866

9. https://www.linkedin.com/pulse/dimensionality-reduction-pca-vs-lda-face-recognition-m-farhan-tandia

10. THÈSE DE DOCTORAT Présentée par : Khalid CHOUGDALI Discipline : Sciences de l'Ingénieur Spécialité : Informatique et Microélectronique, Conception de nouveaux algorithmes pour la reconnaissance des visages