ENSSAT
LANNION

Projet noté

Machine Learning & traitement d'image : les Eigenfaces

colinleverger [at] gmail [dot] com

Colin Leverger - ENSSAT Informatique, Multimédia et Réseaux
Promotion 2017

Destinataire : Claude CARIOU

20 mai 2016

Table des matières

1 Introduction **2**

2 Définition des Eigenfaces **2**

3 Définition du besoin **2**
3.1 Création de bases de données . . . 3
3.1.1 Le jeu de données . . . 3
3.1.2 Que stocker en base ? . . . 3
3.2 Utilisation de la base de données : la reconnaissance de visage . . . 3
3.3 Schémas fonctionnels . . . 5
3.3.1 L'apprentissage . . . 5
3.3.2 Le test de l'algorithme . . . 6

4 Développement : la structure du code **6**

5 Développement : l'apprentissage **7**
5.1 Charger les images . . . 7
5.2 La normalisation . . . 7
5.3 Création des Eigenfaces . . . 8
5.4 Calculs des descripteurs . . . 9

6 Développement : test de l'apprentissage **9**
6.1 Chargement et normalisation de l'image de test . . . 9
6.2 Calcul des descripteurs de l'image . . . 9
6.3 Comparaison des descripteurs stockés et des descripteurs calculés . . . 10

7 Développement : benchmarks et Overall Accuracy **11**
7.1 Le principe . . . 11
7.2 Le résultat des benchmarks . . . 11

8 Améliorations possibles **14**

9 Conclusion **14**

Table des figures **16**

Références **16**

1 Introduction

Dans le cadre de la formation Informatique, Multimédia et Réseaux dispensé à l'ENSSAT de Lannion, nous avons étudié le traitement d'image en seconde année. Cette matière, qui s'inscrit dans la composante «multimédia» de notre cursus, nous a permis de comprendre les concepts clés de la manipulation d'images par programmation. Nous avons notamment travaillé à l'utilisation des Eigenfaces pour reconnaître les visages dans une photo.

Pour pratiquer et appréhender au mieux les sujets évoqués en CM, nous avons effectué un TP noté. Il s'agissait d'écrire avec Matlab un programme permettant de reconnaître un individu en créant et utilisant une base de données.

Le code associé à ce compte rendu pourra être trouvé dans l'archive jointe. Très peu de code sera directement exposé dans le rapport, mais les structures complexes et les choix d'implémentation des algorithmes y seront clairement explicités.

2 Définition des Eigenfaces

En tant qu'être humain, nous pouvons décrire une image de différentes manières : qualité de la photo, présence de couleur ou de niveaux de gris... Nous sommes naturellement capables de distinguer les détails et de reconnaître les objets et personnes présentes sur une photo. L'objectif des Eigenfaces est de donner à l'ordinateur la capacité de reconnaître des visages comme l'être humain pourrait le faire.

Pour décrire une image, un ordinateur va la décomposer en vecteurs descriptifs, que l'on appellera eigenvecteurs. Il s'agit de calculer les valeurs propres associées aux photos considérées ; les eigenvecteurs en découleront. L'ordinateur va stocker les eigenvecteurs pour chaque image et les utiliser a posteriori pour les comparaisons.

Pour expliciter la méthode des Eigenfaces en une phrase, il nous faut capturer les variations dans une collection de photos de visages et utiliser cette information pour encoder et comparer des visages d'individus de manière holistique (autrement dit, l'image vue comme un ensemble de paramètres) [ZT08]

Pour utiliser les Eigenfaces, les informations suivantes seront extraites du jeu d'image :

- Descripteur : résumé d'une l'image, contient des informations résumées sur l'image considérée. Ils sont obtenus en faisant une projection de l'image sur les eigenvecteurs.
- Eigenface : c'est un condensé de toute la base d'image.
- Visage moyen : représente une moyenne de tous les visages mis en base. Utilisé pour normaliser les images fournies à l'algorithme.
- Écart type des visages : comme le visage moyen, cette information sera elle aussi utilisée pour les normalisations.

3 Définition du besoin

La détection de visages par Eigenfaces s'appuie sur deux étapes. Il s'agit dans un premier temps de créer une base de données à partir d'un set d'images. Dans un second temps, il s'agit de comparer les données tirées de l'image en entrée du programme à cette base de données. Seront explicitées dans cette partie ces deux étapes.

3.1 Création de bases de données

3.1.1 Le jeu de données

Disposer d'un jeu de donnée était un prérequis important pour commencer le développement. Celui utilisé ici est une référence dans le monde du traitement d'image, et pourra être trouvé sur le site www.cl.cam.ac.uk. Ici, 10 photos par individu seront disponibles, pour un total de 40 individus. Ce seront donc 400 photos directement exploitables pour créer les Eigenfaces.

Les photos fournies représentent des visages des personnes, prises sous différentes orientations. Elles font une taille de 112 x 92, et n'auront pas besoin d'être retouchées pour les traitements. En d'autres termes, il n'y a pas besoin de les recadrer pour obtenir un visage. Un exemple de 5 images pour l'individu n°1 pourra être trouvé en figure 1 en page 3.

FIGURE 1 – 5 premières images pour l'individu n°1

Les images fournies seront par contre deux fois trop grandes pour les traitements que nous voulons faire, et il sera nécessaire de les redimensionner en divisant leur taille par 2.

Pour la plupart des photos du jeu de données, la position des yeux et des bouches est normalisée. Dans un cas idéal, les yeux et les bouches de chacune des personnes devraient être positionnés au même endroit sur toutes les photos.

3.1.2 Que stocker en base ?

Un certain nombre de données doivent être stockées, à savoir :

- Pour les comparaisons :
  - Eigenfaces : ensemble des eigenvecteurs.
  - d : descripteurs des images.
- Pour les normalisations :
  - m : visage moyen des visages de la base.
  - S : écart type des visages de la base.

Le choix a été fait ici de stocker ces valeurs sous forme de tableaux CSV.

Notez que pour obtenir ces valeurs, un certain nombre de calculs vont être appliqués à nos images. Ces calculs seront explicités dans la partie développement et réalisation des algorithmes.

3.2 Utilisation de la base de données : la reconnaissance de visage

Ici, il s'agit de vérifier le bon fonctionnement de l'algorithme d'apprentissage développé. Pour ce faire, une image qui n'aura pas été mise en base sera fournie à l'algo, et l'image la plus proche trouvée en base devra être retournée.

Par exemple, si nous fournissons l'image n°6 pour l'individu n°1, avec 5 images par individus mises en base, si l'algorithme est précis, il devra retourner comme voisin le plus proche une image

de l'individu n°1.

On observera souvent une différence entre le résultat observé et le résultat attendu. Autrement dit, l'algorithme n'est pas parfait, et plusieurs paramètres peuvent entrer en compte et créer un biais, à savoir :

— Qualité et normalisation des images pas toujours idéale,
— Calculs avec des valeurs tronquées sur certains vecteurs.

C'est pour cette raison qu'un benchmark complet de l'algorithme a été effectué lors de ce TP. Ce benchmark est en fait un «Overall Accuracy». Le choix a été fait de vérifier le retour de l'algorithme et de faire des pourcentages, les taux de réussite. Des graphes seront explicités à la fin de ce rapport.

3.3 Schémas fonctionnels

Avant de commencer le développement, il était important de poser sur le papier les schémas fonctionnels. Ils seront utilisés comme base pour coder les fonctions et algorithmes. Il est d'ailleurs conseillé de se référer à ces deux schémas lors de la partie Développement (voir section 5)

3.3.1 L'apprentissage

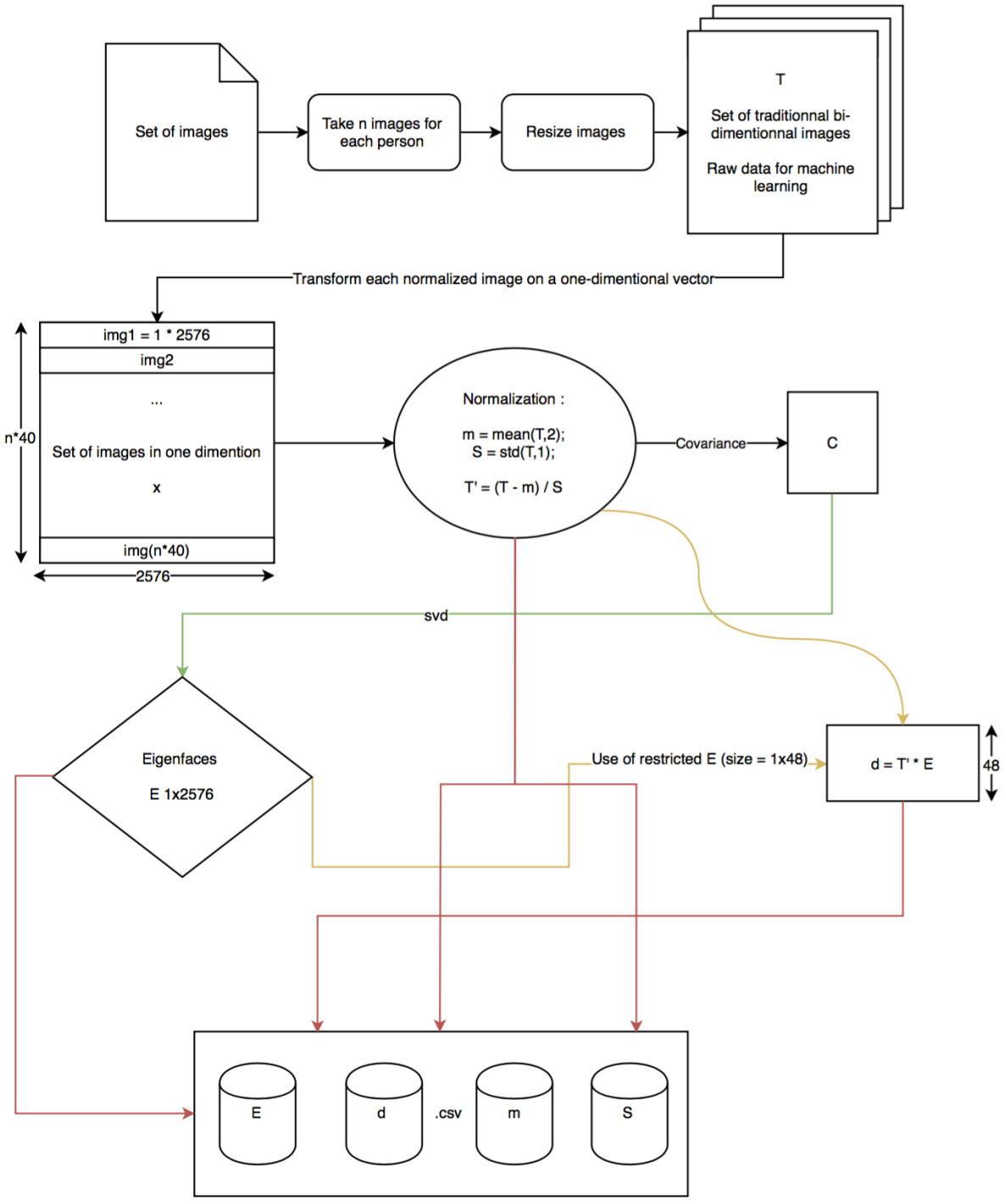

FIGURE 2 – Apprentissage, schéma fonctionnel

3.3.2 Le test de l'algorithme

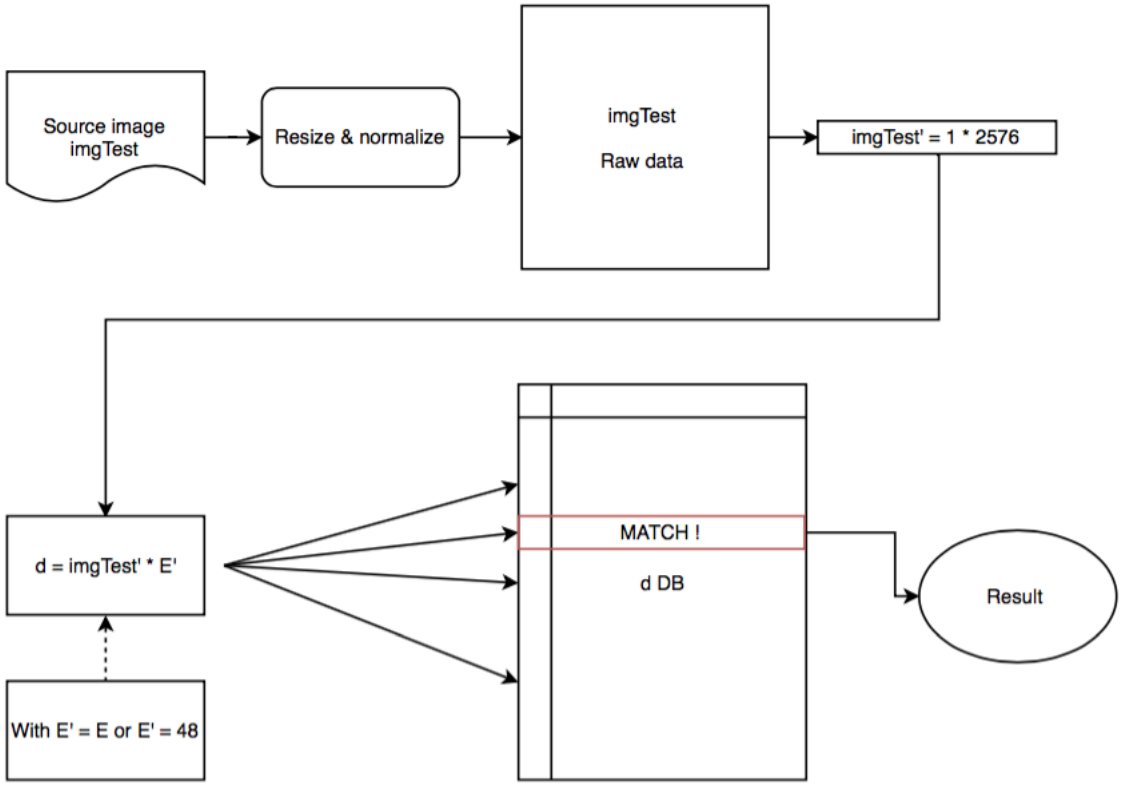

FIGURE 3 – Test de l'algo, schéma fonctionnel

4 Développement : la structure du code

Lors du développement avec Matlab, il est conseillé de décomposer chaque fonctionnalité en une fonction et en un fichier.

Les fichiers seront organisés selon l'arborescence suivante :

- Le dossier «./att_faces/» contiendra les images.
- Le dossier «./data/» contiendra les données extraites des images, stockées lors de la phase d'apprentissage.
- Le dossier «./helpers/» contiendra les scripts utilitaires pour effectuer des opérations mathématiques ou logiques sur les matrices et/ou fichiers.
- Le dossier «./learning/» contiendra les fonctions utilitaires propres à l'apprentissage (calcul de moyenne, des eigenfaces...)
- Le dossier «./results/» contiendra des captures d'écrans et résultats de diverses opérations.
- Le dossier «./scripts/» contiendra trois scripts : «*do_learning*», «*test_learning*» et «*benchmark*», ces trois scripts permettant de tester les algorithmes. Ils vont donc lier et chaîner tous les scripts précédemment cités.
- Le fichier «./overall_accuracy.m» va permettre de lier les trois scripts du dossier «./tests/» pour automatiser les tests/benchmarks des scripts sur plusieurs valeurs différentes de E, plu-

sieurs nombres d'images chargées, etc.

5 Développement : l'apprentissage

5.1 Charger les images

Dans un premier temps, il s'agit de charger en mémoire les images dont nous avons besoin pour effectuer les traitements et calculs.

Une image peut être représentée comme une matrice à deux dimensions, chaque case (*i,j*) de la matrice représentant un pixel. Dans le cas d'une image en noir et blanc, les valeurs seront situées entre 0 et 255 (0 représentant le noir, 255 le blanc).

Le nombre d'images chargées est dynamique et dépendra ici d'un paramètre fixé par l'utilisateur.

Étapes suivies pour charger le set d'images à traiter :

1. Faire un «directory listing» du path fourni par l'utilisateur. Ce dernier représente l'emplacement du jeu de données.
2. Pour chaque dossier trouvé, effectuer un nouveau directory listing afin de trouver les paths de toutes les images de chaque dossier (40 dossiers / 10 images par dossiers)
3. Récupérer les paths de *n* images par dossiers.
4. Pour chaque path :
   (a) Charger l'image associée en réduisant sa taille.
   (b) Mettre l'image chargée sous la forme d'un vecteur. Il faut concaténer toutes les lignes de l'image en une seule et unique ligne. Autrement dit, passer d'une image en deux dimensions (*x,y*) en une image en une dimension (*1,*$x * y$).
   (c) Ajouter ce vecteur à un tableau en deux dimensions (chaque ligne représentant une image)

Lors du «directory listing» de Matlab, les dossiers Unix «.» et «..» sont affichés ; cela pose problème, car il ne faut pas itérer sur ces dossiers, sinon l'algorithme serait sans fin ! Pour résoudre ce problème, une petite fonction utilitaire a été codée afin de les supprimer des résultats.

Lors du chargement des images, deux options ont étés considérées. Les images vont pouvoir être chargées de manière totalement aléatoire ou de manière non aléatoire, en suivant le classement naturel des fichiers (*fichier1.jpg*,*fichier2.jpg*,...,*fichierN.jpg*). On verra lors de la partie benchmarks que ces deux types de chargement offrent des résultats de reconnaissance assez différents. La fonction qui charge les images va donc prendre comme argument un booléen «*random*» et va retourner la liste des images chargées. Cette dernière sera importante pour éviter de tester les images déjà mises en base !

Pour récupérer les images, et les charger en tant que matrices, Matlab propose des fonctions natives telles que *imread*. Ces librairies sont très pratiques, testées et fonctionnelles, elles seront utilisées ici.

Le tableau d'images en ligne chargé sera à partir d'ici appelé T.

5.2 La normalisation

Une fois les images chargées sous la forme d'un tableau, la matière première est prête à être exploitée et nous pouvons commencer les calculs mathématiques. Mais d'abord, une étape de normalisation des images s'impose, afin de travailler sur la même base pour chacune des images.

Pour la normalisation, il s'agit d'effectuer les trois étapes suivantes :

- Calcul de la moyenne des images : pour calculer la moyenne des images (et donc avoir le visage moyen de notre base), il s'agit de calculer la moyenne de chaque pixel en colonne, et de stocker le résultat. Il faut lire chaque colonne de T pour effectuer sa moyenne. Dans l'exemple visible en figure 4 en page 8, on modélise le cas d'une image 3px par 3px. Si on s'intéresse à la première colonne en jaune, sa moyenne sera $(1 + 4 + 4)/3 = 3$ et le résultat sera stockés dans la ligne «*mean*». Un exemple de visage moyen pourra être trouvé en figure 5 en page 8. Notez que lors de cette étape, et pour éviter d'avoir un clash de taille de matrice, il faut utiliser la fonction *repmat* pour les calculs.
- Calcul de l'écart-type de T sur le même principe.
- Créer T2, qui sera notre tableau T après normalisation. T2 sera le résultat du calcul suivant : $T2 = (T - m)./S;$. On note qu'il est important d'utiliser l'opérateur «./» pour la division, car on doit utiliser un calcul point par point et non un calcul matriciel standard.

	1	2	3	2
	4	5	4	3
	4	5	6	2
mean	3	4	4,33333333	2,33333333

FIGURE 4 – Calcul de la moyenne

FIGURE 5 – Exemple de visage moyen

T2 sera utilisé pour le reste des calculs.

5.3 Création des Eigenfaces

L'étape suivante est l'une des plus importantes, puisqu'elle consiste en la création des fameuses Eigenfaces. Pour ces calculs, il faut :

- Calculer la matrice de covariance C de T2. Elle représente les relations et points de similarités entre les différentes images composant T2.
- Calculer la décomposition en valeurs singulières de la matrice de covariance précédemment trouvée. C'est ce qui va créer E. Pour voir un exemple d'eigenfaces, se référé à l'image 6 en page 9.

— Retourner E tronqué à la taille souhaitée pour les exécutions. On s'est en effet rendu compte qu'il n'était pas nécessaire d'utiliser le vecteur E dans sa totalité pour avoir des résultats satisfaisants, et qu'il suffisait souvent de prendre environ les 50 premières images. Un focus sera fait sur ce point dans la partie benchmarks.

FIGURE 6 – Exemple d'Eigenfaces calculé (image tronquée, car l'orignale est trop longue — voir orignale dans l'archive de code, dossier results.)

5.4 Calculs des descripteurs

Les descripteurs s'obtiennent très facilement lorsqu'on a déjà calculé les Eigenfaces. Il s'agit de faire une projection de l'image T2 sur les Eigenfaces. En d'autres termes, il faut faire le calcul $d = T2 * E;$.

Une fois que les descripteurs ont étés calculés, il n'y a plus besoin de faire de calculs, toutes les données dont nous avons besoin sont disponibles. Il ne faut pas oublier de les stocker en base pour pouvoir les réutiliser. Pour rappel, chaque donnée est stockée sous forme d'un tableau CSV.

6 Développement : test de l'apprentissage

Une fois l'apprentissage effectué, il était important d'effectuer différents tests afin de vérifier le retour de l'algo. Cette partie est plus simple que l'apprentissage, et les étapes suivies seront explicités ci-dessous. On notera que toutes les valeurs calculées lors de l'apprentissage seront chargées en début de script et disponibles sous forme de variables Matlab.

6.1 Chargement et normalisation de l'image de test

Nous souhaitons charger une seule image pour voir le retour de l'algorithme. Il faut la normaliser exactement de la même manière que les images que nous avons chargé dans l'apprentissage. Les moyenne et écart type ont d'ailleurs étés stockés durant l'apprentissage pour pouvoir être réutilisés ici.

Pour un rappel sur la normalisation de l'image, se référer à l'étape 3 de la partie normalisation 5.2 en page 7, en prenant en compte qu'ici, c'est une seule image qui est normalisée. T sera donc à une seule dimension (*1,y*). Il faut aussi prendre en compte qu'il n'y a pas besoin de calculer la moyenne et l'écart type (ça n'aurait pas vraiment de sens avec une seule image chargée !), mais plutôt utiliser les valeurs stockées.

6.2 Calcul des descripteurs de l'image

Pour calculer les descripteurs de l'image, il suffit de projeter l'image de test normalisée aux Eigenfaces, exactement de la même manière qu'en partie 5.4 en page 9.

6.3 Comparaison des descripteurs stockés et des descripteurs calculés

La partie la plus intéressante de l'algorithme de test se trouve ici. Pour simplifier, il s'agit de faire une comparaison entre les descripteurs calculés lors de l'apprentissage et les descripteurs calculés avec l'image de test chargée. Une soustraction permettra de trouver la plus petite valeur, et donc l'image la plus proche de celle donnée en argument.

Pour aller dans le détail et expliciter les étapes :

- Faire une soustraction entre les descripteurs de l'image et les descripteurs stockés en base.
- Calculer la distribution euclidienne entre les deux descripteurs. Pour ce faire, effectuer une multiplication de la différence par sa transposée, $diagonal = diag(d2 * transpose(d2))$;.
- Trier cette distribution euclidienne dans l'ordre croissant afin de trouver la première photo la plus proche à l'index 1 du tableau.
- Trouver le plus proche voisin et l'afficher (ou afficher ces 4 plus proches voisins par exemple).

Dans la solution implémentée, une photo aura forcément un voisin le plus proche, même si les deux photos n'ont aucun rapport. Autrement dit, si une photo de camion est testée, il n'y a pas de seuil qui permette de dire si cette photo représente un visage ou non, et un individu sera retourné quoiqu'il arrive.

Pour retrouver le voisin le plus proche (et donc sa classe), il a été ici fait le choix de supposer que les images sont chargées dans l'ordre des dossiers dans la base (à savoir dossier1/, dossier2/, ..., dossierN/). Pour retrouver la classe de l'image considérée, il suffit alors de diviser l'index de l'image trouvée en base par le nombre d'images chargées en base. Exemple : l'image reconnue est la 55e, et nous avons chargé 5 images par classe. La classe sera donc $55/5 = 11$.

Il aurait été possible de mettre au début de chaque vecteur image dans T2 le numéro de sa classe de l'image considère. Cette solution aurait même été plus propre et plus rigoureuse, car cela aurait permis de charger les photos dans n'importe quel ordre.

À la fin de cette opération, nous pouvons (ou non, paramétré par un booléen) afficher les plus proches voisins, comme sur les figures 7 ou figure 8, respectivement en pages 10 et 10.

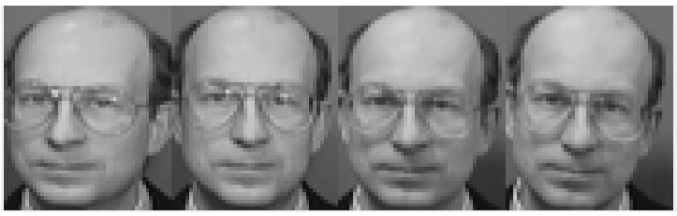

FIGURE 7 – Exemple 1 de résultat retourné par l'algorithme

FIGURE 8 – Exemple 2 de résultat retourné par l'algorithme

Comme vous pouvez le voir sur la figure 8, le résultat n'est pas toujours excellent, et les voisins les plus proches ne sont pas toujours les bons. Ceci peut s'expliquer par plusieurs facteurs, tels que :

— La position de la bouche et des yeux n'est pas la même sur toutes les photos d'un individu.
— Un être humain aurait déjà du mal à reconnaître cet individu...

7 Développement : benchmarks et Overall Accuracy

7.1 Le principe

Afin d'expliquer les différents résultats faux et d'observer les performances de l'algorithme après plusieurs itérations et exécutions, le choix a été fait d'exécuter les processus décrits dans la partie 5 et 6 plusieurs fois en faisant varier différents paramètres.

Les paramètres qui seront modifiés lors du benchmark seront les suivants :

— E : la taille de la troncature de E utilisée va varier de 10 à 100.
— Le nombre d'images chargé en base de données va varier de 1 à 9. On s'attend à obtenir des résultats excellents lorsque 9 images seront chargées.
— Le chargement aléatoire ou non des images lors de la création de T.

Pour chaque modification de paramètre, des tests seront effectués pour chaque image non chargée en base. La fonction «*test_learning*» renvoie un résultat, qui n'est autre que la classe du premier candidat identifié par l'algo. Comme nous connaissons (grâce à une regex sur le nom du path) la classe de l'image que nous testons, il s'agit de vérifier le retour et d'incrémenter la matrice de confusion. La matrice de confusion, dans la terminologie de l'apprentissage supervisé, est un outil servant à mesurer la qualité d'un système de classification [Wik16]. Ici, il s'agit d'une matrice de 40x40. En ordonnée, il y a les classes détectées et en abscisse, les classes réelles de l'image considérée. Donc, à chaque itération, on incrémente la matrice de confusion à l'indice (*class_of_image,class_found*).

À la fin des exécutions (autrement dit, quand on a testé toutes les images qui n'avaient pas étés chargées en base), on récupère la diagonale de cette matrice et on observe le pourcentage de bonne reconnaissance. En effet, la diagonale de la matrice de confusion représente les indices (*class_of_image,class_found*) quand ces deux valeurs sont égales : c'est une bonne reconnaissance ! Pour obtenir des pourcentages, il suffit de diviser les valeurs de la diagonale par la somme de toutes les valeurs de la matrice. Un exemple de matrice de confusion parfaite (avec 100% de bonnes reconnaissances) pourra être trouvé en image 9 en page 12.

Il est à noter que l'exécution des benchmarks pour les différentes valeurs variables cités ci-dessus prend beaucoup de temps : sous un Mac de 2015 double-cœur, cela prend en effet plus de 3 heures...

7.2 Le résultat des benchmarks

Les figures 10 et 11 représentent les résultats de l'overall accuracy en modifiants plusieurs paramètres. Ces screenshots seront d'ailleurs présents dans l'archive de ce rapport en meilleure qualité.

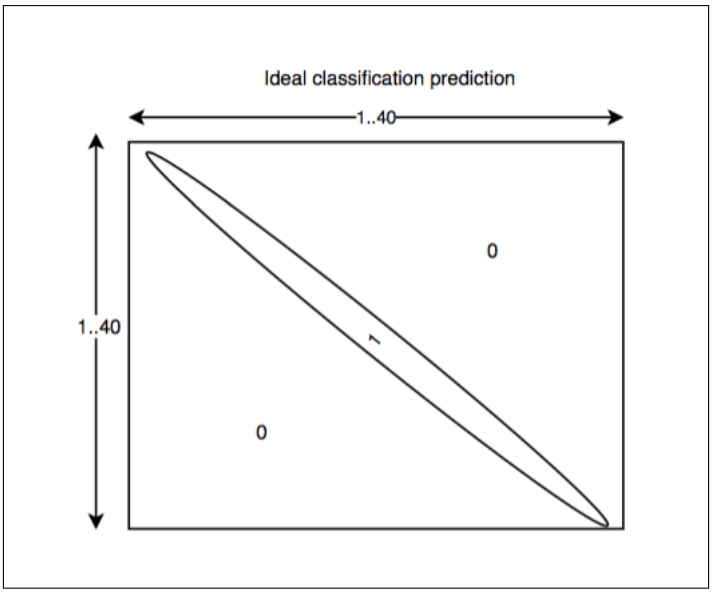

FIGURE 9 – Matrice de confusion parfaite

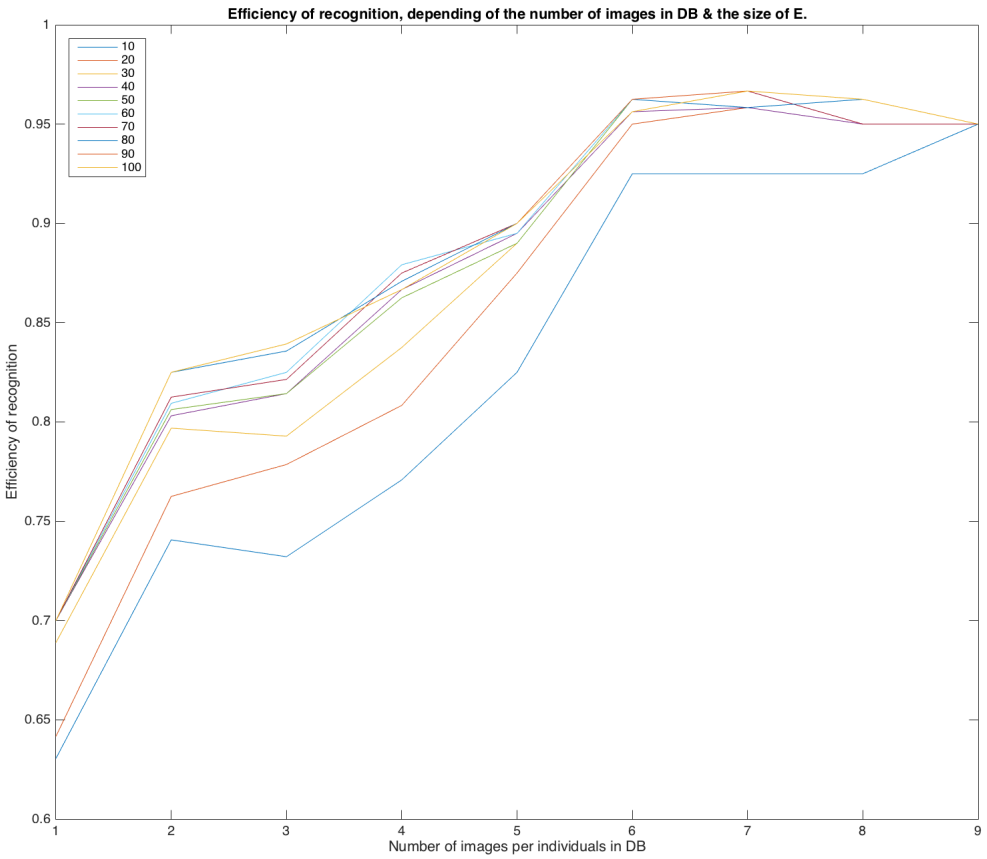

FIGURE 10 – Images chargées de manière non aléatoire dans la base

Nous pouvons déduire de l'étude de la figure 10 que :

— Lorsque les images sont chargées dans l'ordre, l'algorithme n'est pas précis à 100%, mais à

95% seulement.
— On observe facilement qu'augmenter la taille tronquée de E n'a pas de grosse incidence après 50, comme expliqué plus haut dans ce rapport.
— Au-delà d'un certain seuil d'image chargée en base, visible ici à 6 images, il n'est plus tellement intéressant en termes de qualité de reconnaissance d'en ajouter d'autres.

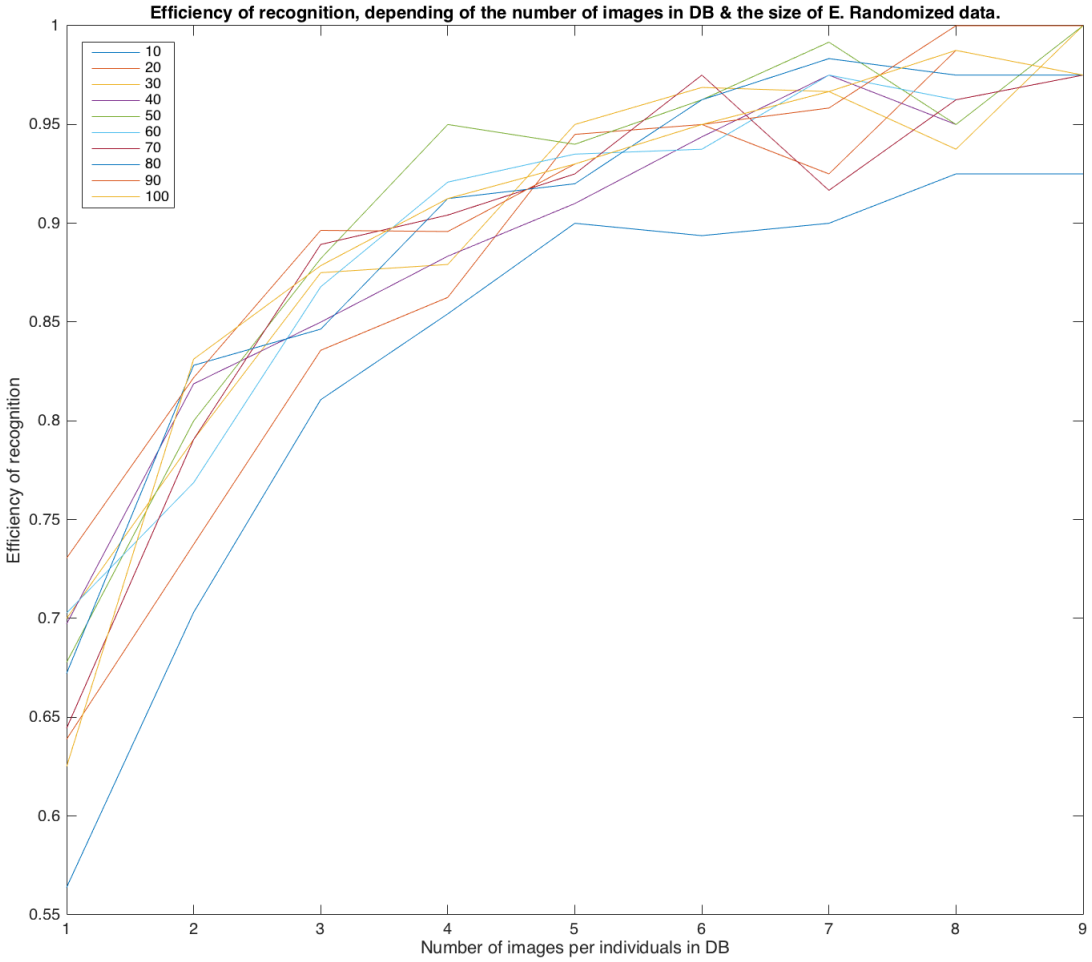

FIGURE 11 – Images chargées de manière aléatoire dans la base

Nous pouvons déduire de l'étude de la figure 11 que :

— Lorsque les images ne sont pas chargées dans l'ordre, l'algorithme est dans l'ensemble plus précis et arrive plusieurs fois à un taux de reconnaissance de 100
— Même constat sur la taille de E.
— Dans l'ensemble, l'efficacité des reconnaissances suit une progression plus linéaire.

On peut en conclure que l'ajout d'un caractère aléatoire lors du chargement d'une photo permet d'avoir des résultats plus probants et doit être une piste à privilégier lors du développement.

8 Améliorations possibles

Ce projet était très intéressant, mais faute de temps, un bon nombre de features n'ont pas pu être développées. Les améliorations possibles (et souhaitées) pourraient être les suivantes :

- Rendre le chargement des photos interactif et totalement indépendant du nommage des dossiers. Pour l'instant, les dossiers sont bien triés : ils débutent avec le préfixe «s» et sont suivis du numéro de la classe (1,2,...,40). De plus, le path doit être fourni en dur directement dans le code, ce qui rend toute modification plus complexe. Fournir une interface utilisateur pour paramétrer ce path pourrait être pratique, et inscrire un marqueur de classe au début de chaque photo du tableau T serait aussi une solution plus robuste.
- Pouvoir régler les options (chargement «random» des photos, affichage ou non des résultats...) pourrait être sympathique pour l'utilisateur.
- Affiner l'apprentissage et pouvoir choisir entre les 3 voisins les plus proches celui qui serait le plus approprié, basé sur des paramètres de précédent apprentissage. Renforcer l'aspect incrémental de l'apprentissage.
- Détecter de manière automatique si une photo représente un visage. Pour ce faire, il aurait été possible de coder selon deux manières :
  - Fixer un seuil empirique pour déterminer si une photo est un visage. Par exemple, avec une distance calculée de plus de 6000 pour le voisin le plus proche, décréter empiriquement que cette photo n'est pas un visage. L'inconvénient de cette méthode est que le seuil empirique doit être fixé par l'utilisateur, et est donc sujet aux erreurs...
  - Fixer un seuil dynamiquement. Pour fixer le seuil, il faudrait :
    1. Reconstruire l'image testée à partir des eigenfaces,
    2. Faire une soustraction entre l'image reconstruite et l'image donnée,
    3. Vérifier si les images ne sont pas trop différentes.
- Tester chacune des fonctions, afin de suivre une méthodologie Test Driven Dévelopement. Le code fourni serait alors plus facilement maintenable, mais la TDD ne s'applique pas facilement à Matlab...
- Donner la possibilité à l'utilisateur de lier le projet à une base MySQL pour rendre la base d'apprentissage plus facilement réutilisable hors contexte.
- Tester l'algorithme avec plus de photos, représentant des visages ou non.

J'espère avoir le temps après cette période-école de continuer à développer plusieurs features cités ci-dessus.

9 Conclusion

J'ai réussi à développer l'apprentissage et le test de la base assez rapidement. Cela m'a permis de développer des tests plus poussés, notamment l'overall accuracy/benchmark, et je suis content d'avoir pu observer l'influence des paramètres (taille de E, nombre d'images en base...) sur les taux de reconnaissance. Il était aussi surprenant d'observer qu'après certains seuils, augmenter le nombre d'images en base détériorait les résultats au lieu de les améliorer. Après des recherches sur internet, j'ai appris que l'on appelait ce phénomène le «suraprentissage» : le modèle a tellement appris sur les données fournies qu'il a perdu en généralisation. L'analyse fine de ce genre de comportement était pour moi un aspect très intéressant.

Ce projet nous a permis d'appréhender un bon nombre de concepts propre au machine learning :

création de descripteurs, stockage en base, utilisation de l'apprentissage... Cette première vision des possibilités énormes offerte par le machine learning m'a donné envie d'approfondir et de creuser d'avantage ces concepts. Bien que l'utilisation des mathématiques peut parfois compliquer hautement les opérations, l'expérience était très plaisante et j'espère pouvoir évoluer dans ce domaine lors de mes prochaines expériences scolaires et professionnelles.

S'il fallait refaire ce projet, j'aurais certainement essayé d'utiliser un langage de programmation différent (tel que le Scala, par exemple) et j'aurais aussi essayé de faire varier le jeu de donnée (cas de personnes non caucasiennes...). L'utilisation de Matlab était tout de même très intéressante et ce langage me semble très puissant pour ce genre de projets.

Le code associé à ce projet pourra être trouvé sur GitHub, à l'adresse suivante : https://github.com/ColinLeverger/eigenfaces-matlab.

Table des figures

1 5 premières images pour l'individu n°1 . . . 3
2 Apprentissage, schéma fonctionnel . . . 5
3 Test de l'algo, schéma fonctionnel . . . 6
4 Calcul de la moyenne . . . 8
5 Exemple de visage moyen . . . 8
6 Exemple d'Eigenfaces calculé (image tronquée, car l'orignale est trop longue — voir orignale dans l'archive de code, dossier results.) . . . 9
7 Exemple 1 de résultat retourné par l'algorithme . . . 10
8 Exemple 2 de résultat retourné par l'algorithme . . . 10
9 Matrice de confusion parfaite . . . 12
10 Images chargées de manière non aléatoire dans la base . . . 12
11 Images chargées de manière aléatoire dans la base . . . 13

Références

[Wik16] Wikipedia : Matrice de confusion. (2016)

[ZT08] Zhang, S. ; Turk, M. : Eigenfaces. 3 (2008), Nr. 9, S. 4244. – revision 128015