Reconnaissance de visages
INF552 Projet

Sébastien Dubois, Alexandre Sablayrolles

Janvier 2015

Contents

1 Introduction

Ce projet, réalisé dans le cadre du cours INF552, a pour but de créer un programme de reconnaissance de visages (*face recognition*) capable de s'entraîner sur une base de données de visages.

Initialement, le sujet proposé était la reconnaissance d'objets dans des images avec OpenCV et le modèle du bag-of-words. Nous avons choisi d'appliquer cette méthode plus spécifiquement à la reconnaissance de visages. Nous avions pour objectif de télécharger des images de célébrités sur Google Image et d'entraîner notre algorithme sur ces images. Pour des raisons techniques, nous n'avons pu télécharger suffisamment d'images et avons donc opté pour la Yale Face Database [2]. Nous avons également supposé que ces images contenaient toutes exactement un visage. Cette hypothèse est notamment importante pour les images qui servent à entraîner l'algorithme, pour lesquelles nous pourrons toujours supposer qu'il y a exactement un visage. Pour le cas général (plusieurs visages dans une image), il suffirait de modifier la méthode d'application de l'algorithme (mais pas la méthode d'entraînement) en conséquence.

Dans ce rapport, nous différencions deux catégories d'images. Afin de constuire un modèle et de le tester, nous répartissons en effet notre base de données en deux parties : les données *training*, qui servent à entraîner l'algorithme, et les données *test*, qui simulent des images non-étiquetées.

2 Présentation des différentes méthodes

2.1 Représentation des visages

Dans cette partie, nous décrivons les différentes approches avec lesquelles nous avons tenté de représenter les visages, afin de pouvoir en extraire des caractéristiques déterminantes permettant de classifier les images, et de prédire ultérieurement le nom d'un visage inconnu.

Comme cela a été vu en cours, représenter une image seulement par les intensités des pixels qui la composent est une approche qui comporte plusieurs failles. On citera par exemple l'orientation de l'image et l'exposition lumineuse. De ce fait, nous nous sommes naturellement intéressés aux descripteurs SIFT (Scale-Invariant Feature Transform [1],[4]), qui représentent les directions locales du gradient tout en essayant de s'affranchir de l'échelle et de l'orientation.

2.1.1 Extraction des SIFTs et approche Bag-Of-Words

La première méthode que nous avons implémentée s'inspire des méthodes générales de reconnaissance d'objets. Il s'agit de représenter une image par la liste des SIFTs qui la composent. Ainsi, on estime la proximité sémantique de deux images par le nombre de SIFTs qu'elles ont en commun.

Afin de structurer cette approche et de limiter l'espace mémoire nécessaire, on effectue en réalité une première étape de clustering sur l'ensemble des images

étiquetées : c'est la méthode des *Bag-of-Words*.

Etape 1 :
Traiter toutes les images dans *training*.
Extraire tous les SIFTs pour chaque image.

Etape 2 :
Déterminer k centres de clusters, où k est fixé. Cela fournit le *dictionnaire*.

Etape 3 :
Pour toute image dans *training*, calculer son histogramme sur le dictionnaire. Pour cela, chaque descripteur SIFT de l'image est identifé au plus proche centre de cluster.

Etape 4 :
Associer à toute image, dans *training* ou *test*, son histogramme associé au dictionnaire, afin de construire un classifieur (SVM par exemple) ou d'utiliser ce classifieur.

Cette première méthode peut apporter des résultats satisfaisants mais cela nécessite de considérer des dictionnaires de taille importante. Par ailleurs, cette approche ne prend aucun *a priori* sur les images qu'elle traite. On pourrait donc l'utiliser pour différencier également d'autres types d'objets. Cependant elle n'exploite pas la structure des visages et c'est de cette façon que nous avons souhaité améliorer notre algorithme.

2.1.2 Clustering des zones du visage

Dans cette deuxième méthode nous avons donc voulu exploiter la structure des visages, en ne calculant que les descripteurs SIFTs d'un visage sur des zones bien précises. Grâce aux classifieurs *HaarCascade* [3] fournis avec OpenCv, il est effectivement possible de repérer certaines zones du visage, telles que les yeux, le nez, la bouche, etc ...

Notre idée était donc de construire des *Bag-of-Words* non pas sur l'ensemble des SIFTs mais plutôt de construire un *Bag* par zone. Etant donné une nouvelle image, on pourrait alors estimer, pour chaque zone, à quel élément du dictionnaire celle-ci s'identifie. Bien que cette méthode s'inspire des *Bag-of-Words*, représenter les images par leur histogramme n'a pas fourni de résultats satisfaisants. Cependant l'approche classique de clustering s'est montrée plus convaincante. Il s'agit de construire un dictionnaire, de taille fixée, pour chaque zone. Etant donné une nouvelle image, on l'identifie comme la liste des éléments du dictionnaire auxquels elle correspond (et non plus à un histogramme). On dirait par exemple que cette image est représentée par [Bouche n°3, Nez n°10, Oeil Gauche n°1, Oeil Gauche n°5].
Enfin nous construisons des classifieurs pour chaque zone, basés sur cette représen-

tation.

Pour calculer un descripteur SIFT, il faut spécifier une taille et un angle. Nous déterminons la taille du SIFT par rapport à la taille de la zone detectée avec les classifieurs *HaarCascade*. L'orientation du visage est en général plus difficile à déterminer, et pourtant essentielle.

Nous avons décidé de renseigner l'orientation d'un visage par l'angle formé entre l'horizontale et l'alignement des deux yeux. Ainsi, nous pouvons calculer cet angle lorsque les deux yeux sont detectés. Dans le cas contraire, nous pouvons aussi estimer cet angle lorsque nous détectons le nez et la bouche en considérant que cette droite est perpendiculaire à l'alignement des yeux.

2.1.3 Classification dans l'espace des SIFTs

Cette seconde méthode nous a conduit à nous affranchir des dictionnaires pour travailler directement dans l'espace des SIFTs, soit $\mathbb{R}^{128}$. Afin de réduire la dimension d'étude, d'éviter le sur-apprentissage, ou tout simplement d'améliorer les performances de nos algorithmes, nous avons décidé d'utiliser des méthodes de *feature selection* et d'*ACP* (Analyse en Composantes Principales - PCA en anglais). Ces méthodes, ainsi que les manières dont nous les avons utilisées, sont plus largement décrite dans la section 2.3.

Par ailleurs, nous avons omis jusqu'à présent un problème important que soulève la séparation du visages en plusieurs zones : comment ré-unifier l'information obtenue par chaque zone afin d'en tirer des conclusion sur le visage tout entier ? Afin d'illustrer cette problématique, prennons deux exemples.
Exemple 1 : Nous cherchons à identifer Pierre, Paul ou Jacques dans des images. Pierre et Paul ont des bouches qui se ressemblent très fortement, alors que Paul et Jaques ont des yeux presque semblables. Il serait souhaitable que notre programme réponde "Paul" s'il reconnaît sur un même visage la bouche de Pierre et les yeux de Jacques.
Exemple 2 : Si Pierre, Paul et Jacques ont tous les trois une bouche semblable, il serait souhaitable que notre programme tienne peu compte de la bouche pour favoriser les yeux ou le nez.

Cela laisse présumer qu'il serait préférable de considérer un seul descripteur global construit par la juxtaposition des descripteurs de chaque zone, et ce fut notre première approche.

Cependant il est assez fréquent que nous ne détections pas toutes les zones sur une image. Dans ce cas, le descripteur d'une zone non détectée était remplacé par une suite de zéros. Cela introduisait finalement trop d'imprécision et la méthode consistant à classifier une image zone par zone s'est avérée plus performante.

Pour cela, nous exploitons le fait que les classifieurs SVM ne retournent pas une réponse binaire mais un réel. Ces classifieurs seront plus largement détaillés

dans la section 2.3, mais nous pouvons dès à présent noter que le résultat réel d'une classification par SVM permet d'estimer la confiance en cette prédiction. Ainsi, nous calculons le résultat global pour un visage comme la somme des résultats réels prédits pour chaque zone. Cela permet de favoriser les zones pour lesquelles on est *sûrs* de la prédiction.

Par ailleurs nous avons pensé qu'il serait possible d'attribuer des poids à chaque zone (pour multiplier le résultat de prédiction) qui seraient estimés durant le processus d'apprentissage. Cependant nous n'avons pas eu le temps d'implémenter cela dans la durée limitée du projet.

2.2 Amélioration de la détection de zones

La détection de zones (visage, yeux, bouche, etc.) présente certains problèmes dans notre programme. Le premier problème apparaît lorsque trop de zones sont détectées. C'est le cas pour les visages dans certaines images : plusieurs potentiels visages sont détectés et fournis par OpenCV sous la forme d'un tableau, dans lequel les zones sont ordonnées (la meilleure étant la première). Néanmoins, la zone qu'OpenCV estime comme étant le meilleur visage n'est pas toujours le vrai visage, alors que celui-ci fait toujours partie des visages détectés. Nous avons donc choisi d'implémenter une fonction qui sélectionne le meilleur visage parmi une liste de potentiels visages. Pour cela, nous avons utilisé une heuristique simple : le meilleur visage est celui qui contient le plus de zones caractéristiques (bouche, nez, yeux). Cette heuristique est en pratique très bonne (c'est notamment dû au fait que les détecteurs de bouche repèrent bien les yeux!). Cela a permis de détecter 400 visages plutôt que 100 pour la personne B22 de la Yale Face Database.

Un second problème qui émerge est que certaines zones ne sont jamais détectées. Pour certaines images, il est par exemple impossible de détecter des yeux. Pour remédier à ce problème, nous avons implémenté deux améliorations. La première consiste, dans le cas où on ait détecté un oeil, un nez et une bouche, à calculer le symétrique de cet oeil par rapport à l'axe nez-bouche. La deuxième consiste à utiliser des statistiques sur la position moyenne des yeux (voir Figure 1) pour intuiter la position la plus probable des yeux dans le cas où aucun oeil n'est détecté. A cette fin, nous avons enregistré des statistiques sur la position des yeux dans le visage, leur largeur et leur hauteur afin de prédire au mieux ces données lorsqu'elles ne sont pas disponibles.

2.3 Méthodes statistiques de traitement des données

Dans cette partie nous présentons plusieurs méthodes statistiques que nous utilisons dans notre projet. Les deux premières permettent de réduire la dimension et la suivante concerne la classification.

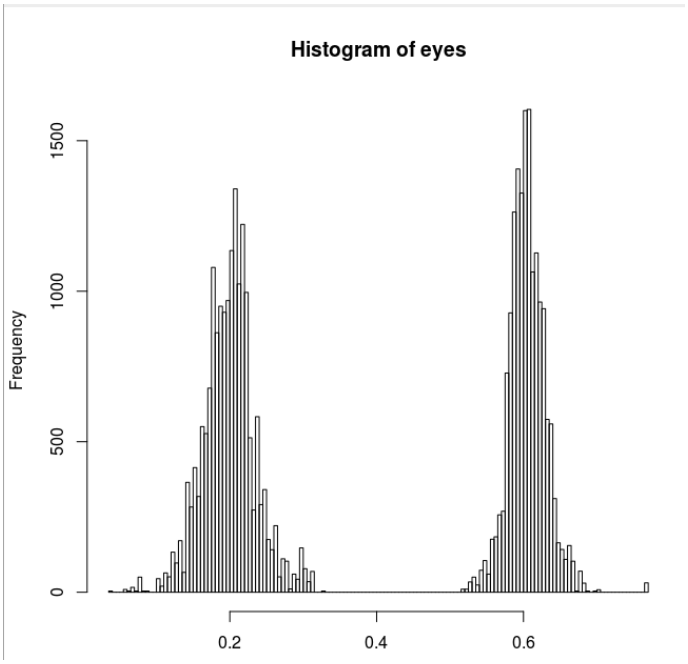

Figure 1: Histogramme de l'abscisse des yeux (ramenée à la largeur du visage)

2.3.1 Feature selection

La *feature selection* consiste à ne garder qu'une partie des coordonnées des vecteurs considérés. L'idée principale est la suivante : sur l'ensemble des vecteurs considérés, certaines coordonnées sont corrélées. Il suffit donc de garder certaines coordonnées "caractéristiques" et de supprimer les coordonnées redondantes (car corrélées aux coordonnées caractéristiques). De plus, certaines données peuvent ne donner que très peu d'information et ainsi introduire du bruit.

Pour appliquer la *feature selection*, nous avons considéré deux méthodes de filtrage : *information gain* et *chi-square test*. Ces méthodes estiment la qualité de chaque coordonnée pour ensuite les ordonner. La méthode d'*information gain* estime le gain d'information apporté par une coordonnée, tandis que la méthode du *chi-square* évalue l'indépendance entre coordonnées. Ces deux méthodes étant différentes, leurs résultats sont donc différents. De plus, comme ce sont des méthodes de filtrage, qui *estiment* la qualité des coordonnées, leur résultat est à nuancer.

Cela permet donc de réduire la dimension de l'espace considéré et éventuellement d'améliorer les performances de la classification grâce à la réduction du bruit. Mais cette projection peut aussi diminuer l'étalement des points dans l'espace. Pour minimiser cette perte d'information, nous avons donc considéré également l'analyse en composantes principales.

2.3.2 ACP

L'ACP (*analyse en composantes principales*) est aussi un procédé de réduction de la dimension qui consiste à projeter les données sur un sous-espace vectoriel de dimension plus petite. Contrairement à la *feature selection*, l'espace considéré dans l'ACP n'est pas nécessairement un sous-ensemble de l'espace des coordonnées considéré.

En effet, l'objectif de l'ACP est de préserver la variance du nuage de points. Lorsque l'on projette en dimension p en effectuant une ACP, on projette donc sur le sous-espace de dimension p qui va maximiser l'étalement des données.

2.3.3 Classifieur SVM

Dans un problème de classification à deux classes, l'une des approches les plus simples pour construire un classifieur est de séparer l'espace en deux avec un hyperplan affine. le SVM (*Support Vector Machine*, machine à vecteurs supports) implémente cette approche. Formellement, un classifieur SVM est constitué d'un vecteur ω et d'un scalaire b. Etant donné un point x de l'espace, ce point appartient à la classe A si et seulement si :

$$\omega x + b \leq 0$$

Plus généralement, dans un problème de classifications multi-classes, on considère une approche *one versus all* : pour chaque classe, on entraîne un classifieur disant si le point appartient ou non à la classe. Si l'on possède C classes, la classe retenue sera :

$$\texttt{argmin}_{1 \leq j \leq C} \omega_j x + b_j$$

2.3.4 Visualisation

Nous avons écrit quelques scripts Python afin de visualiser les données en 3D, notamment pour estimer la bonne répartition des différentes classes et donc la qualité des modèles de représentation du visage.

Le script *feature_selection* représente en 3D les données de chaque zone, avec sélection des coordonnées ou non. De plus, ce script enregistre l'ordre d'importance des coordonnées, déterminé par les méthodes d'*information gain* et du *chi square*.

Le script *eval_pca* représente la conservation de l'information ainsi que celle de la variance en fonction de la dimension de réduction, et ce pour chaque zone.

3 Organisation du code et des fichiers

3.1 Organisation des fichiers

- allFeatures
  - *database*
- build
- classifiers
- data ..... Images d'entraînement et de test
  - *database*
    - labeled ..... Entrainement
    - unlabeled ..... Test
- dictionary
  - *database*
    - dictionary-yml (BOW method)
- lib ..... Fichiers de Cascade d'OpenCV
  - haarcascade-*-xml
- stats ..... Statistiques sur la position des zones
  - eye-x.csv
  - eye-y.csv
  - eye-width.csv
  - eye-height.csv
- viz ..... Scripts python relatifs à la *feature selection*

3.2 Organisation du code

3.2.1 Méthode Bag-of-Words

Tout le code spécifique à la méthode des Bag-of-Words se situe dans le fichier *bowMethod.cpp*.

La fonction *buildBowDictionary* traite toutes les images d'entraînement afin de construire un dictionnaire.
La fonction *createBowClassifier* génère et sauvegarde les classifieurs, grâce au dictionnaire et images d'entraînement, et sauvegarde les histogrammes de celles-ci.
La fonction *computeBowTestDesciptors* génère les histogrammes des images tests.
La fonction *bowPredict* utilise les histogrammes sauvegardés pour tester le modèle sur les classifieurs précédemment calculés.

3.2.2 Fichier featureDetection

Le fichier *featureDetection.cpp* implémente les différentes méthodes basées sur la représentation du visage en zones spécifiques.

La fonction *featureExtraction* traite toutes les images d'entraînement et de test pour en extraire les descripteurs SIFTs. Celle-ci n'a besoin d'être appelée qu'une seule fois pour chaque méthode de détection (cf. partie suivante).
La fonction *clusteringClassifyAndPredict* implémente la méthde de *clustering des zones du visage* (cf. partie 2).
Les fonctions *classifyAndPredictSingleDescriptor* et *classifyAndPredict* implémentent la classification dans l'espace des SIFTs, la première utilisant un seul descripteur construit par concaténation des descripteurs de chaque zone.
Toutes les fonction *Predict* testent le modèle ainsi construit.

Les méthodes de détection :
Il est possible d'utiliser 3 méthodes différentes de détection de zones pour ces fonctions.
La méthode *simple* : c'est la méthode de base.
La méthode *bestFace* : on estime le *meilleur* visage dans une image en comptant le nombre de bouches, nez et yeux que chaque éventuel visage détecté contient (cf. 2.2).
La méthode *completed* : elle utilise non seulement la méthode bestFace mais elle intuite également les yeux manquants lorsque cela est possible (cf. 2.2).

3.2.3 Autres

Fichier faceDetection :
Ce fichier contient toutes les fonctions particulières à la detection de zones du visage.

Fichier getSiftKeypoints :
Ce fichier contient les fonctions déterminant les *keypoints* SIFTs, en fonction de chaque zone. On utilise ici la structure connue du visage pour améliorer cette détection. Par exemple, on ne recherche la bouche que dans la moitié basse du visage.

Fichier tools:
Ce fichier contient quelques fonctions auxiliaires.

3.2.4 Guide d'utilisation

Les images sur lesquelles on souhaite travailler doivent se situer dans le dossier /data/*database*, où *database* correspond au nom de la base de données avec laquelle on travaille, et doivent être réparties en deux dossiers, labeled (pour l'entraînement) et unlabeled (pour les tests).

Il suffit d'appeler la fonction *featureExtraction* une seule fois (ce qui a déjà été fait pour la Yale Face Database).
Il est ensuite possible de tester les différentes méthodes en choisissant les paramètres souhaités : dimension de l'ACP, nombre de coordonnées sélectionnées,

méthode de détection, validation croisée ou non (pour la construction des classifieurs).
De plus, des constantes globales permettent de spécifier le nombre de personnes différentes apparaissant dans les images de la base de données et si l'on souhaite effectuer l'étape de sélection de coordonnées ou non.

Pour l'approche de Bag-of-Words, une seule méthode est disponible. Il est nécessaire d'appeler les trois fonctions *buildBowDictionary*, *createBowClassifier* et *computeBowTestDesciptors* avant de pouvoir effectuer des tests (ce qui a déjà été fait pour la Yale Face Database).

Comme les fonctions d'extraction de descripteurs sont assez lentes, ceux-ci ont déjà été calculés et sauvegardés. Cependant il peut être intéressant d'appeler la fonction *featureExtraction* avec l'argument *verbose = true* afin de visualiser la détection de zones. Si cette opération est effectuée, il est conseillé de spécifier un argument *detectionType* $\notin \{0,1,2\}$ afin de ne pas compromettre les sauvegardes précédentes.

Par ailleurs, il semblerait que certains bugs subsistent dans OpenCV lors du chargement de descripteurs Bag-of-Words et de classifieurs SVM en *Release mode*. Pour cette raison, il faut compiler en *Debug mode* pour exécuter sans erreur les fonctions : *classifyAndPredict, classifyAndPredictSingleDescriptor, createBowClassifier, computeBowTestDesciptors* (les deux premières s'executent malgré tout très rapidement).

Les fonctions permettant de tester les modèles (classification et prédiction) possèdent plusieurs paramètres pouvant être modifiés à discrétion. L'argument *nb_components* règle la dimension d'ACP et doit être inférieurs au nombre de coordonnées selectionnées, qui est déterminé par l'argument *nb_features*. Ceux-ci doivent également être inférieur à 128 (qui est la taille d'un descritpeur SIFT). Il est également possible de choisir la méthode de détection de zones : 0 pour *simple*, 1 pour *bestFace* et 2 pour *completed* (cf. 3.2.2). Il est enfin possible de choisir si l'on souhaite effectuer une validation croisée lors de la construction des classifieurs via l'argument *cross_valid*.

4 Résultats

Méthode	Taux d'erreur d'entraînement	Taux d'erreur de test
Bag of Words	B20: 14% sur 513 images B21: 83% sur 529 images B22: 30% sur 535 images	B20: 8% sur 25 images B21: 67% sur 28 images B22: 34% sur 29 images

Table 1: Résultats du Bag-of-Words avec 50 clusters

Clustering	Taux d'erreur d'entraînement	Taux d'erreur de test
nb_clusters = 50	B20 : 42% sur 466 images B21 : 43.0155% sur 451 images B22 : 49.676% sur 463 images	B20 : 30% sur 23 images B21 : 38% sur 26 images B22 : 45% sur 24 images
nb_clusters = 100	B20 : 39% sur 466 images B21 : 45% sur 451 images B22 : 38% sur 463 images	B20 : 17% sur 23 images B21 : 42% sur 26 images B22 : 62% sur 24 images
nb_clusters = 150	B20 : 33.691% sur 466 images B21 : 33.4812% sur 451 images B22 : 31.1015% sur 463 images	B20 : 21% sur 23 images B21 : 46% sur 26 images B22 : 29% sur 24 images

Table 2: Résultats du clustering avec *selectBestFace* et sans cross Validation

Descripteur unique	Taux d'erreur d'entraînement	Taux d'erreur de test
features = 128 PCA = 128	B20 : 40% sur 466 images B21 : 29% sur 451 images B22 : 15% sur 463 images	B20 : 21% sur 23 images B21 : 34% sur 26 images B22 : 16% sur 24 images
features = 128 PCA = 60	B20 : 23% sur 466 images B21 : 24% sur 451 images B22 : 21% sur 463 images	B20 : 8% sur 23 images B21 : 30% sur 26 images B22 : 12% sur 24 images
features = 100 PCA = 40	B20 : 18% sur 466 images B21 : 31% sur 451 images B22 : 59% sur 463 images	B20 : 4% sur 23 images B21 : 34% sur 26 images B22 : 70% sur 24 images

Table 3: Concaténation des descripteurs

Classification par zone	Taux d'erreur d'entraînement	Taux d'erreur de test
features = 128 PCA = 128	B20 : 21% sur 466 images B21 : 9% sur 451 images B22 : 9% sur 463 images	B20 : 30% sur 23 images B21 : 19% sur 26 images B22 : 8% sur 24 images
features = 128 PCA = 60	B20 : 25% sur 466 images B21 : 4% sur 451 images B22 : 3% sur 463 images	B20 : 47% sur 23 images B21 : 11% sur 26 images B22 : 4% sur 24 images
features = 100 PCA = 40	B20 : 42% sur 466 images B21 : 25% sur 451 images B22 : 32% sur 463 images	B20 : 43% sur 23 images B21 : 19% sur 26 images B22 : 45% sur 24 images

Table 4: Séparation des classifieurs par zone

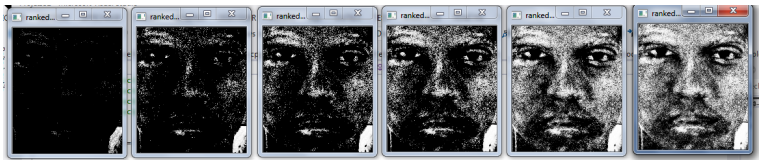

Figure 2: Pixels les plus discriminants, méthode du *chi square*

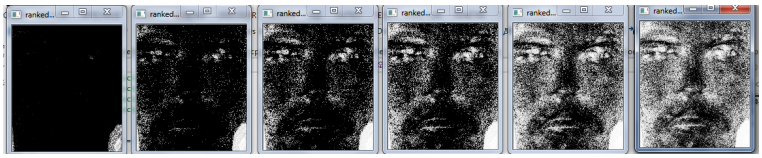

Figure 3: Pixels les plus discriminants, avec *information gain*
De gauche à droite : le nombre de coordonnées selectionnées est augmenté.
Importance des coordonnées : du plus clair au plus foncé.

5 Pour aller plus loin

Lors du développement de notre projet, nous avons eu de nouvelles idées que nous n'avons pas pu implémenter par manque de temps ou par volonté de ne pas disperser notre analyse.

Les premières pistes de réflexion que nous aurions pu explorer concernent la détection de zones. Nous pourrions par exemple chercher des yeux, bouches et nez partout dans l'image et décider que lorsque ces trois attributs sont "compatibles", le rectangle contenant ces attributs forme un visage. De manière générale, certains nouveaux algorithmes performants de détection de zones (fonctionnant par exemple avec des réseaux de neurones) ne sont pas implémentés sur OpenCV, ce qui diminue la qualité des résultats espérés.

D'autres pistes autour de la représentation des images auraient pu aboutir à des résultats différents. En premier lieu, nous aurions pu calculer un SIFT global sur le visage de la personne et l'ajouter aux autres descripteurs. De plus, nous avions pensé à chercher plus précisément quelles sont les zones discriminantes d'un visage et qui pourraient ainsi apporter de meilleurs résultats à notre projet, plutôt que de considérer des zones telles que les yeux ou la bouche. Pour illustrer cela, nous avons effectué une *feature selection* des pixels des visages. Les résultats obtenus apparaissent sur les figures 2 et 3.

Enfin, d'un point de vue statistique, nous avons souhaité garder une approche simple mais il est tout à fait envisageable de développer l'étude en considérant d'abord d'autres noyaux, et même d'autres classes de classifieurs.

6 Conclusion

En conclusion, nous avons réussi au cours de ce projet à développer plusieurs méthodes de reconnaissance de visages, largement testées sur une base de données, nous confrontant à des visages d'orientations et d'expositions variées.

Les différentes techniques sur lesquelles nous avons travaillé montrent de nettes différences de performances. En particulier, les premières méthodes de *clustering* sont moins convaincantes.

Par ailleurs, il nous a semblé essentiel d'exploiter au maximum la structure connue du visage, afin d'améliorer nos algorithmes, notamment en se concentrant sur des zones spécifiques (yeux, bouches, nez). Les résultats obtenus montrent bien la pertinence de cette approche. De plus, nous avons pu améliorer la détection de zones justement en tenant compte de cette structure, et ainsi compléter les classifieurs *HaarCascade* pour perfectionner la détection de visage.

Nous pensons ainsi que cette approche pourrait être encore développée en continuant dans cette voie, notamment pour intuiter certaines zones manquantes du visage, car notre présente méthode *completed* est encore peu convaincante. Nous pensons également qu'il serait intéressant de considérer d'autres zones du visages, déterminées par une étude statistique.

References

[1] *Distinctive Image Features from Scale-Invariant Keypoints*. URL: http://www.cse.unr.edu/~bebis/CS491Y/Papers/Lowe04.pdf.

[2] *Extended Yale Face Database B (B+)*. URL: http://vision.ucsd.edu/content/extended-yale-face-database-b-b.

[3] *Haar Feature-based Cascade Classifier for Object Detection*. URL: http://docs.opencv.org/modules/objdetect/doc/cascade_classification.html.

[4] *OpenCV - Introduction to SIFT*. URL: http://docs.opencv.org/master/doc/py_tutorials/py_feature2d/py_sift_intro/py_sift_intro.html.