INSTITUT DE LA FRANCOPHONIE POUR L'INFORMATIQUE (IFI)

****

RAPPORT FINAL DU PROJET

Rédigé par : **BIAKOTA BOMBIA Herbert Cephas**

MILORME Pierre Rubens

Plan

Introduction :

le monde actuel dans lequel nous sommes , l'image, qui est d'une part un signal 2D (x,y) qui représente souvent, une réalité 3D (x,y,z) et d'autre part du point de vue humain, celle-ci contient plusieurs informations sémantiques qu'il faudrait interpréter son contenu au-delà de la valeur des nombres, est omniprésente à la télévision, dans des magazines, dans la presse en général et sur l'Internet . Cependant, Il serait à l'heure actuelle difficile de se passer d'images. Ainsi, l'accumulation d'images numériques soulève une problématique très importante dans domaine de recherche d'images car l'approche la plus ancienne et encore majoritairement utilisé aujourd'hui est l'annotation textuelle telle que les mots-clés, Pour ce fait, dans le cas des bases d'images de grandes tailles, la description textuelle pour toutes les images de la base est une opération longue, coûteuse et pénible pour l'utilisateur. Ainsi, l'un des inconvénients de cette recherche par mot clés dépend de la langue utilisée et que celle -ci peut donner des résultats complètement hors sujet comme par exemple le mot '**avocat**' qui désigne un homme de loi et en même temps un fruit. Afin d'échapper à de telles situations, une solution consiste à éviter la procédure de l'utilisation des mots-clés et donc de passer à une autre technique qui prend en considération l'image et uniquement l'image pour effectuer les recherches. Cette méthode est dite la **Recherche d'Images par le Contenu** (RIC ou CBIR, Content Based Image Retrieval). Ainsi, La recherche d'images par le contenu est devenue un domaine de recherche très actif depuis plusieurs années maintenant c'est dans ce contexte que notre notre travail ce orienté

Première partie : Utilisation de descripteurs globaux sur des images

Une présentation détaillée de votre travail et des méthodes utilisées
le programme que nous avons implémenté fonctionnent sous python en ligne de commande de la manière suivante:

fonctionnent et description du programme
- pour pouvoir exécuter notre programme, nous devons à partir de ligne commande nous positionner dans le répertoire de notre Nom_Fichier.py et taper la syntaxe suivante : «**python NomFichier.py -f ImageRequête -M ValeurHistogramme -N NombreDeKnn** ».
Ensuite notre programme doit tout d'abord calculer les histogrammes pour chaque image se trouvant dans notre base d'image ainsi que celui de l'image requête tout en les normalisant. Chacun de ces histogrammes(image de la base et image requête) sera convertis en vecteur à une dimension et calculer ensuite la distance (distance d'euclidienne) entre chaque vecteur d'image de la base et celui de l'image requête. Après ce traitement, notre programme va,

grâce à la valeur de réduction de l'histogramme en vue d'optimiser en temps de calcul, nous ressortir les K plus proches voisins de manière croissante avec avec leur distance par rapport à l'image requête c'est à dire les N images les plus similaires.Ex: « python knn -f obj1__340.png -M 8 -N 10

a- Description globale du contenu des images

- L'histogramme : des niveaux de gris ou des couleurs d'une image est une fonction qui donne la fréquence d'apparition de chaque niveau de gris (couleur) dans l'image

b - Système de recherche d' images similaires

comme nous mentionné dans la présentation de notre programme tout se fera en ligne de commande et la sortie sera affichée sous forme texte vers la sortie standard. En argument (entrée) du programme, il faut spécifier le nom d'un fichier image. Ensuite, le système calcule la distance entre cette image et toutes les images de la base d'images. En sortie, le système affiche les N premières images, c'est-à-dire triées dans l'ordre de distance de la plus petite à la plus grande. Un argument d'entrée permet de spécifier N (par défaut 10). Pour cela, nous utilisons un classifieur de type K-Plus Proches Voisins etnfin predire la objet associée à notre image de requête (obj*)

NB : le texte :(image requete:4426) : Gtk-WARNIING ** : Unable to lovate theme engine in module_path : 'adwaita') qui se sort sur nos captures d'écran est un probleme de graphique

<u>résultat du traitement</u>

Pour test de 8 valeurs par couleurs avec N=5

python knn -f obj1__340.png -M 8 -N 10

```
        Image de requete = obj2__265.png
        Valeur de M = 8
        Valeur de N = 5


I)KNN : K plus proches voisins(K images les plus similaires)


Images                  Distances


(image requete:4426): Gtk-WARNING **: Unable to locate theme engine in module_path: "adwaita",
"obj2__260.png"         0.209085
"obj2__270.png"         0.309932
"obj2__255.png"         0.34379
"obj2__250.png"         0.417515
"obj2__245.png"         0.49447

Determination de la classe de l'image requete (classe majoritaire des ): 5 plus proches voisins
        [(5, 'obj2')]
La classe a laquelle appartient l'image est : obj2
```

fig1 : classe d'image de requête est obj2

Pour test de 8 valeurs par couleurs avec N = 10

python knn -f obj2__265.png -M 8 -N 10

```
        Image de requete = obj2__265.png
        Valeur de M = 8
        Valeur de N = 10


I)KNN : K plus proches voisins(K images les plus similaires)


Images                  Distances


(image requete:4596): Gtk-WARNING **: Unable to locate theme engine in module_path: "adwaita",
"obj2__260.png"         0.209085
"obj2__270.png"         0.309932
"obj2__255.png"         0.34379
"obj2__250.png"         0.417515
"obj2__245.png"         0.49447
"obj2__240.png"         0.533271
"obj2__235.png"         0.582377
"obj2__275.png"         0.597257
"obj2__230.png"         0.642644
"obj2__280.png"         0.653977

Determination de la classe de l'image requete (classe majoritaire des ): 10 plus proches voisins
        [(10, 'obj2')]
La classe a laquelle appartient l'image est : obj2
```

fig2 : classe d'image de requête est obj2

Pour test de 16 valeurs par couleurs avec k = 5

python knn -f obj2__265.png -M 16 -N 5

```
        Image de requete = obj2__265.png
        Valeur de M = 16
        Valeur de N = 5


I)KNN : K plus proches voisins(K images les plus similaires)


Images                  Distances


(image requete:4653): Gtk-WARNING **: Unable to locate theme engine in module_path: "adwaita",
"obj2__260.png"         0.482283
"obj2__270.png"         0.633241
"obj2__255.png"         0.786759
"obj2__250.png"         0.91223
"obj2__245.png"         1.01242

Determination de la classe de l'image requete (classe majoritaire des ): 5 plus proches voisins
        [(5, 'obj2')]
La classe a laquelle appartient l'image est : obj2
```

fig3: classe d'image de requête est obj2

Pour test de 128 valeurs par couleurs avec N = 10

python knn -f obj2__265.png -M 16 -N 10

```
        Image de requete = obj2__265.png
        Valeur de M = 16
        Valeur de N = 10


I)KNN : K plus proches voisins(K images les plus similaires)


Images                  Distances


(image requete:4708): Gtk-WARNING **: Unable to locate theme engine in module_path: "adwaita",
"obj2__260.png"         0.482283
"obj2__270.png"         0.633241
"obj2__255.png"         0.786759
"obj2__250.png"         0.91223
"obj2__245.png"         1.01242
"obj2__240.png"         1.05319
"obj2__235.png"         1.10363
"obj2__275.png"         1.14339
"obj2__280.png"         1.23026
"obj2__230.png"         1.24839

Determination de la classe de l'image requete (classe majoritaire des ): 10 plus proches voisins
        [(10, 'obj2')]
La classe a laquelle appartient l'image est : obj2
```

fig4: classe d'image de requête est obj2

Pour test de 24 valeurs par couleurs avec N =5

python knn -f obj2__265.png -M 24 -N 5

```
        Image de requete = obj2__265.png
        Valeur de M = 24
        Valeur de N = 5


I)KNN : K plus proches voisins(K images les plus similaires)


Images                  Distances


(image requete:4751): Gtk-WARNING **: Unable to locate theme engine in module_path: "adwaita",
"obj2__260.png"         0.998846
"obj2__270.png"         1.12516
"obj2__255.png"         1.38526
"obj2__250.png"         1.553
"obj2__245.png"         1.65858

Determination de la classe de l'image requete (classe majoritaire des ): 5 plus proches voisins
        [(5, 'obj2')]
La classe a laquelle appartient l'image est : obj2
```

fig5: classe d'image de requête est obj2

Pour test de 24 valeurs par couleurs avec N =10

python knn -f obj2__265.png -M 24 -N 10

```
        Image de requete = obj2__265.png
        Valeur de M = 24
        Valeur de N = 10


I)KNN : K plus proches voisins(K images les plus similaires)


Images                  Distances


(image requete:4803): Gtk-WARNING **: Unable to locate theme engine in module_path: "adwaita",
"obj2__260.png"         0.998846
"obj2__270.png"         1.12516
"obj2__255.png"         1.38526
"obj2__250.png"         1.553
"obj2__245.png"         1.65858
"obj2__240.png"         1.87513
"obj2__235.png"         1.88237
"obj2__275.png"         1.96433
"obj2__230.png"         2.04088
"obj2__280.png"         2.1069

Determination de la classe de l'image requete (classe majoritaire des ): 10 plus proches voisins
        [(10, 'obj2')]
La classe a laquelle appartient l'image est : obj2
```

fig6: classe d'image de requête est obj2

Pour test de 24 valeurs par couleurs avec N =10

python knn -f obj2__265.png -M 24 -N 10

```
        Image de requete = obj2__265.png
        Valeur de M = 24
        Valeur de N = 100


I)KNN : K plus proches voisins(K images les plus similaires)


Images                  Distances


(image requete:4902): Gtk-WARNING **: Unable to locate theme engine in module_path: "adwaita",
"obj2__260.png"         0.998846
"obj2__270.png"         1.12516
"obj2__255.png"         1.38526
"obj2__250.png"         1.553
"obj2__245.png"         1.65858
"obj2__240.png"         1.87513
"obj2__235.png"         1.88237
"obj2__275.png"         1.96433
"obj2__230.png"         2.04088
"obj2__280.png"         2.1069
"obj2__225.png"         2.33818
"obj2__220.png"         2.44451
"obj2__215.png"         2.46579
"obj2__285.png"         2.49699
"obj2__210.png"         2.52598
"obj2__290.png"         2.63661
"obj2__125.png"         2.66049
"obj2__195.png"         2.70892
"obj2__150.png"         2.72474
"obj2__120.png"         2.7541
"obj2__205.png"         2.84881
"obj2__135.png"         2.86935
"obj2__180.png"         2.87071
"obj2__160.png"         2.87372
"obj2__130.png"         2.88176
"obj2__95.png"          2.89566
"obj2__200.png"         2.90336
"obj2__165.png"         2.92405
"obj2__115.png"         2.92481
```

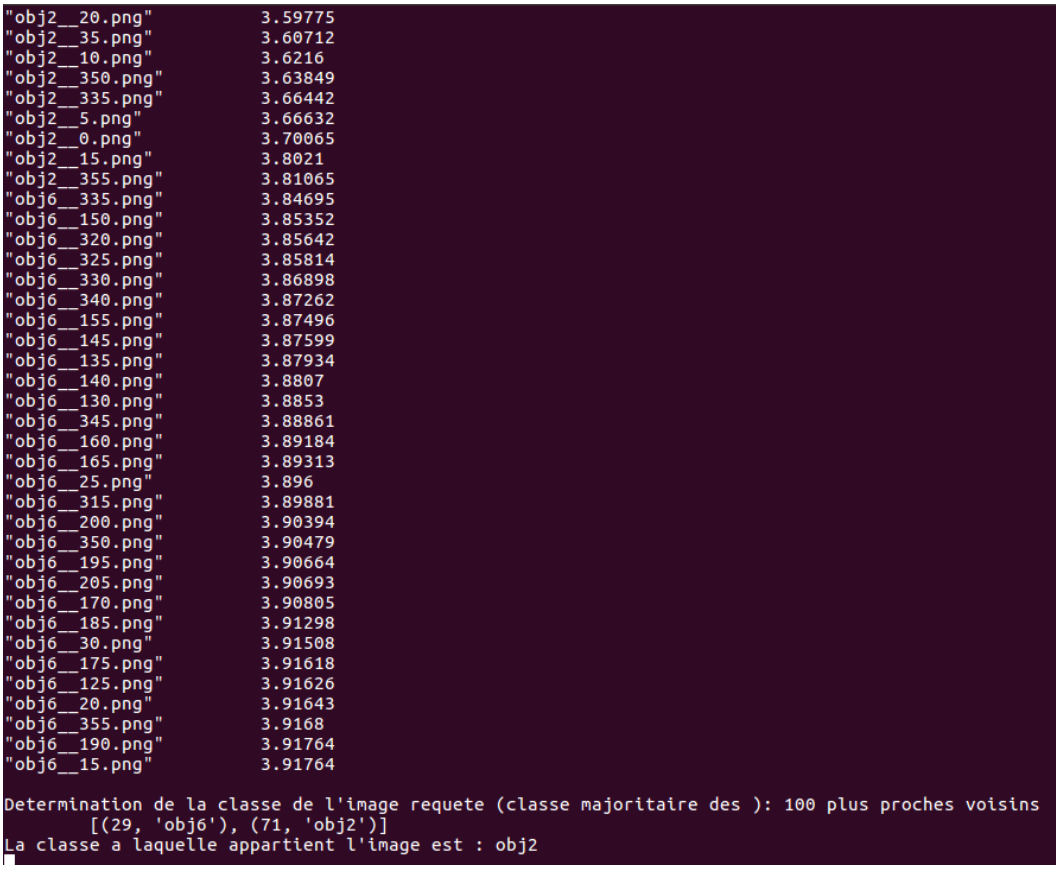


fig7: classe d’image de requête est obj2

- **Moments Hu**

fonctionnent et description du programme

- pour pouvoir exécuter notre programme, nous devons à partir de ligne commande nous positionner dans le répertoire de notre Nom_Fichier.py et taper la syntaxe suivante : «**python fichier.py f1 Image1 f2 image2 -T valeurDequantification**». Ex : python moment_de_Hu.py -f1 bird16.pgm -f2 bird02.pgm -T 8 .Pour ce fait, notre programme doit tout d’abord :

- convertir les images en image en niveau de gris :Nous allons calculer les moments de Hu uniquement sur l'image en niveaux de gris pour simplifier la tâche

- Réduction de l’image en niveau de gris

En ce qui concerne les couleurs et les textures, il n'est pas nécessaire de garder tous les niveaux de gris dans le calcul des textures. Nous allons réduire la quantification de l'image pour passer de 256 niveaux de gris à T niveaux de gris (T = 8, 16 ou 24)

- calcul de moment de Hu
nous calculons le moment sur les formules suivante ;

Basic moments

$$m_{pq} = \int_{-\infty}^{+\infty}\int_{-\infty}^{+\infty} I(x,y)x^p y^q dxdy \quad p \in \bullet^+, q \in \bullet^+$$

Centralized moments

$$M_{pq} = \int_{-\infty}^{+\infty}\int_{-\infty}^{+\infty} I(x,y)(x-x_0)^p (y-y_0)^q dxdy \quad p \in \bullet^+, q \in \bullet^+$$

Normalized moments

$$\mu_{pq} = \frac{M_{pq}}{M_{00}^{1+\frac{p+q}{2}}} \quad aussi\ noté\ \frac{M_{pq}}{M_{00}^{\gamma}}\ avec\ \gamma = \frac{p+q}{2}+1$$

$$Dist(im1,im2) = \frac{\sqrt{\sum_{i=1}^{7}(param_i(im1) - param_i(im2))^2}}{7}$$

Résultat du traitement
python moment_de_Hu.py -f1 bird16.pgm -f2 bird02.pgm -T 8

```
7.27074671242e-06
```

fig:8

python moment_de_Hu.py -f1 bird16.pgm -f2 bird02.pgm -T 24

```
2.10382783805e-05
```

fig:9

3. F ONCTION GLOBALE DE SIMILARITÉ ENTRE DEUX IMAGES

A l’issue du descripteur couleur et des moments de Hu, afin de pouvoir combiner ces deux descripteurs, nous allons définir une fonction globale de similarité entre deux images qui consiste en la somme pondérée de ces deux distances (distance couleur et distance de moment:
Distance(globale)= poids *Distance(couleur) + (1-poids) * Distance(momentsHu)
où poids prend une valeur entre 0 et 1 (valeur à passer en argument de notre programme).
fonctionnent et description du programme
- pour pouvoir exécuter notre programme, nous devons à partir de ligne commande nous positionner dans le répertoire de notre Nom_Fichier.py et taper la syntaxe suivante : «**python fichier.py f Image1 -M valeurDequantification -N NombreDeKnn -poids [0;1]** ». Ex : python distance_global.py -f obj3__135.png -M 8 -N 5 -poids 0.5. Pour ce fait, le respect de cette syntaxe est oblige. Le programme que nous avons implémenté calcule la distance globale des distances couleur et celle de moment de Hu tout en faisant intervenir le poids qui est compris entre 0 et 1.

voir code en annexe(code descripteur globale)

résultat du traitement
python distance_global.py -f obj3__135.png -M 8 -N 5 -poids 0.5.

```
        Image de requete = obj3__135.png
        Valeur de M = 8
        Valeur de N = 5


I)KNN : K plus proches voisins(K images les plus similaires)


Images                  Distances


(image requete:11357): Gtk-WARNING **: Unable to locate theme engine in module_path: "adwaita",
"obj3__140.png"         0.0458391897416
"obj3__125.png"         0.0862994612936
"obj3__150.png"         0.130170574165
"obj3__155.png"         0.158017730923
"obj3__115.png"         0.184581505285

Determination de la classe de l'image requete (classe majoritaire des ): 5 plus proches voisins
        [(5, 'obj3')]
La classe a laquelle appartient l'image est : obj3
```

fig9 : avec classe d’obj3

python distance_global.py -f obj2__265.png -M 8 -N 5 -poids 0.5.

```
        Image de requete = obj2__265.png
        Valeur de M = 8
        Valeur de N = 5


I)KNN : K plus proches voisins(K images les plus similaires)


Images                  Distances


(image requete:11418): Gtk-WARNING **: Unable to locate theme engine in module_path: "adwaita",
"obj2__230.png"         0.321342087962
"obj2__175.png"         0.370903512263
"obj2__185.png"         0.386689182571
"obj2__300.png"         0.515315153109
"obj2__345.png"         0.526407639833

Determination de la classe de l'image requete (classe majoritaire des ): 5 plus proches voisins
        [(5, 'obj2')]
La classe a laquelle appartient l'image est : obj2
```

fig10 : avec classe d'obj2

python distance_global.py -f obj3__135.png -M 8 -N 5 -poids 0.5.

```
        Image de requete = obj1__145.png
        Valeur de M = 8
        Valeur de N = 5


I)KNN : K plus proches voisins(K images les plus similaires)


Images                  Distances


(image requete:11471): Gtk-WARNING **: Unable to locate theme engine in module_path: "adwaita",
"obj1__140.png"         0.0280589784982
"obj1__155.png"         0.0317976787734
"obj1__160.png"         0.059278220884
"obj1__125.png"         0.0739146109589
"obj1__50.png"          0.100125991569

Determination de la classe de l'image requete (classe majoritaire des ): 5 plus proches voisins
        [(5, 'obj1')]
La classe a laquelle appartient l'image est : obj1
```

fig11 : avec classe d'obj1

python distance_global.py -f obj94__40.png -M 8 -N 5 -poids 0.5.

```
        Image de requete = obj94__40.png
        Valeur de M = 8
        Valeur de N = 5


I)KNN : K plus proches voisins(K images les plus similaires)


Images                  Distances


(image requete:11579): Gtk-WARNING **: Unable to locate theme engine in module_path: "adwaita",
"obj94__45.png"         0.00962449546895
"obj94__30.png"         0.0126443910518
"obj94__35.png"         0.0127239467757
"obj94__70.png"         0.0145160506044
"obj94__50.png"         0.0149604999616

Determination de la classe de l'image requete (classe majoritaire des ): 5 plus proches voisins
        [(5, 'obj94')]
La classe a laquelle appartient l'image est : obj94
```

fig12 : avec classe d'obj94

python distance_global.py -f obj3__135.png -M 8 -N 5 -poids 0.5.

```
        Image de requete = obj89__265.png
        Valeur de M = 8
        Valeur de N = 5


I)KNN : K plus proches voisins(K images les plus similaires)


Images                  Distances


(image requete:11682): Gtk-WARNING **: Unable to locate theme engine in module_path: "adwaita",
"obj89__275.png"                0.0192493965595
"obj89__280.png"                0.023683780202
"obj89__270.png"                0.0289018493091
"obj89__260.png"                0.0302424567386
"obj89__255.png"                0.0316197278145

Determination de la classe de l'image requete (classe majoritaire des ): 5 plus proches voisins
        [(5, 'obj89')]
La classe a laquelle appartient l'image est : obj89
```

fig13 : avec classe d'obj89

4. C LUSTERING

Afin d'accélérer les performances de notre système, nous allons devoir y intégrer un algorithme de clustering qui va vous permettre d'accélérer la phase de recherche. L'une des méthodes les plus simples pour la création de cluster est k-means

Deuxième partie : Utilisation de descripteurs locaux et approche Bag-of-Words

Pour cette deuxième partie, nous allons construire un système parallèle au premier et vous devrez comparer les résultats. Cette seconde approche repose sur l'utilisation de descripteurs locaux à partir d'un processus en 2 étapes:

1. Unepartie apprentissage

2. Une partie recherche où vous allez comparer une image requête au résultat de la phase d'apprentissage

1. P HASE D ' APPRENTISSAGE

A- les points d'intérêt

fonctionnent et description du programme

- pour pouvoir exécuter notre programme, nous devons à partir de ligne commande nous positionner dans le répertoire de notre Nom_Fichier.py et taper la syntaxe suivante : «**python fichier.py f1 Image1** ».

Calculer un descripteur local pour chacun des keypoints

fonctionnent et description du programme

- pour pouvoir exécuter notre programme, nous devons à partir de ligne commande nous positionner dans le répertoire de notre Nom_Fichier.py et taper les syntaxes suivante :

à priory, avant de lancer le programme il faut classifier les images :

voire(code classifié image)

et ensuite :

«**python fichier.py -d chemin_to_folders_with_images**». Ex : python **python apprentissage.py -d base-100/** .Pour ce fait, notre programme doit tout d'abord :

Mettre en place d'un classificateur d'image basé sur le contenu à l'aide du modèle de mots visuels dans Python.

Comme son nom l'indique, ce n'est qu'un exemple minimal pour illustrer le fonctionnement général d'un tel système. Le code n'est pas optimisé pour la vitesse, la consommation de mémoire ou les performances de reconnaissance. Pour une vérification du système plus avancée (état 2008): https://github.com/shackenberg/phow_caltech101.py

Si vous avez besoin de résultats de pointe pour la classification de l'image, vérifiez [keras] [5].

L'approche se compose de deux étapes principales appelées apprentissage et classement, représentées dans les fichiers [`learn.py`] et [` classify.py`].

Le script `apprentissage.py` générera un vocabulaire visuel et formera un classificateur à l'aide d'un jeu fourni par l'utilisateur d'images déjà classées.
Après la phase d'apprentissage, «classify.py» utilisera le vocabulaire généré et le classificateur qualifié pour prédire la classe pour toute image donnée au script par l'utilisateur.

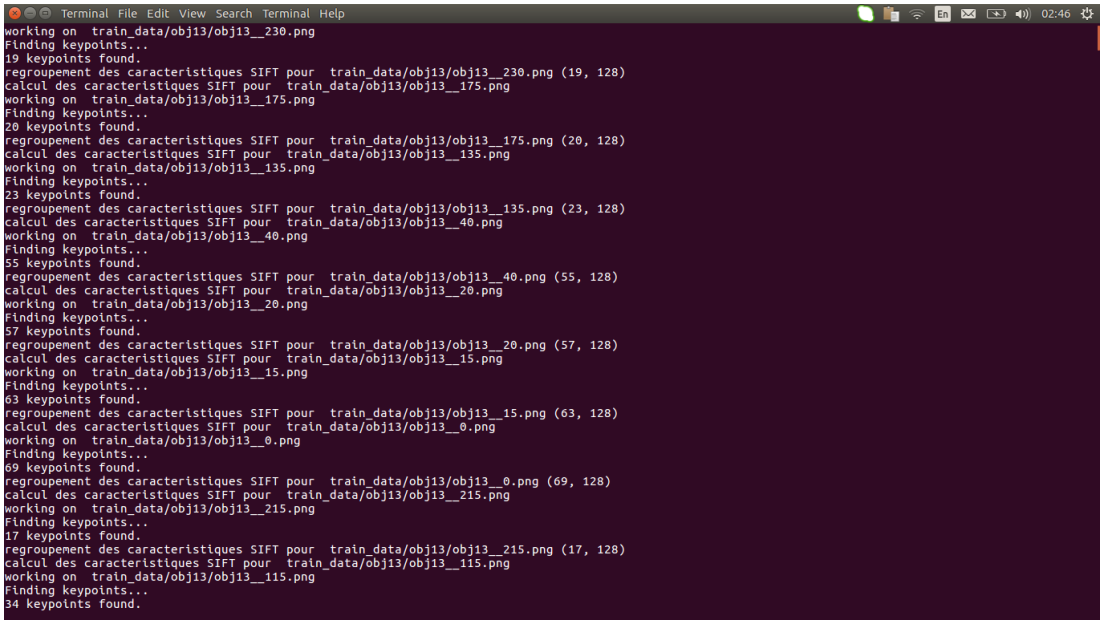
L'apprentissage consiste à:

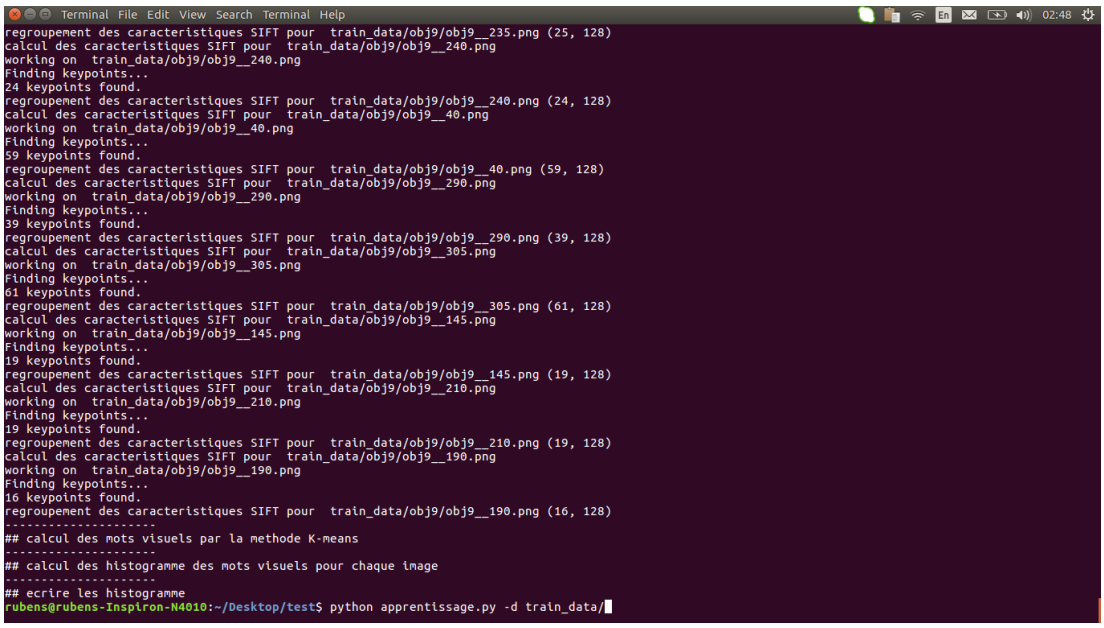
1. Extraction des fonctionnalités locales de toutes les images des jeux de données
2. Générer un codebook de mots visuels avec le regroupement des fonctionnalités
3. Agrégation des histogrammes des mots visuels pour chacune des images traning
4. Alimentation des histogrammes au classificateur pour former un modèle

La classification se compose de:

1. Extraction des caractéristiques locales de l'image classée
2. Agréger les histogrammes des mots visuels pour l'image en utilisant le livre de code généré précédemment
4. Alimentation de l'histogramme au classificateur pour prédire une classe pour l'image

voir en annexe : code **code calcul des descripteur local pour chaque point d'intérêt**





Notre code s'appuie sur:

- Caractéristiques EIP pour les fonctionnalités locales
- k-means pour la génération des mots via le clustering
- SVM comme classificateur utilisant la bibliothèque LIBSVM

Exemple d'utilisation:

Vous formez le classificateur pour un jeu de données spécifique avec:

Python apprentissage.py -d chemin_to_folders_with_images

Pour classer les images, utilisez:

Python classify.py -c path_to_folders_with_images / codebook.file -m path_to_folders_with_images / trainingdata.svm.model images_you_want_to_classify

L'ensemble de données devrait avoir une structure suivante, où toutes les images appartenant à une classe se trouvent dans le même dossier:

```
.
  |-- chemin des repertoires avec les iimages
  |    |-- class1
  |    |-- class2
  |    |-- class3
  ...
  |    └-- classN
```

les histogrammes stockés dans un fichier train_data.svm

0 0:0.000000 1:0.000000 2:0.000000 3:0.000000 4:0.009901 5:0.000000 6:0.009901 7:0.000000 8:0.000000 9:0.000000 10:0.000000 11:0.009901 12:0.000000 13:0.000000 14:0.019802 15:0.000000 16:0.000000 17:0.000000 18:0.000000 19:0.000000 20:0.000000 21:0.000000 22:0.029703 23:0.009901 24:0.000000 25:0.000000 26:0.000000 27:0.000000 28:0.009901 29:0.000000 30:0.000000 31:0.000000 32:0.029703 33:0.000000 34:0.000000 35:0.000000 36:0.000000 37:0.000000 38:0.000000 39:0.009901 40:0.000000 41:0.000000 42:0.000000 43:0.009901 44:0.000000 45:0.000000 46:0.000000 47:0.000000 48:0.000000 49:0.009901 50:0.000000 51:0.000000 52:0.009901 53:0.000000 54:0.000000 55:0.000000 56:0.000000 57:0.000000 58:0.000000 59:0.009901 60:0.009901 61:0.000000 62:0.000000 63:0.009901 64:0.000000 65:0.000000 66:0.029703 67:0.000000 68:0.000000 69:0.000000 70:0.000000 71:0.000000 72:0.000000 73:0.000000 74:0.000000 75:0.000000 76:0.000000 77:0.000000 78:0.000000 79:0.000000 80:0.000000 81:0.000000 82:0.000000 83:0.000000 84:0.000000 85:0.000000 86:0.000000 87:0.000000 88:0.000000 89:0.000000 90:0.000000 91:0.000000 92:0.000000 93:0.000000 94:0.000000 95:0.019802 96:0.009901 97:0.000000 98:0.000000 99:0.000000 100:0.000000 101:0.000000 102:0.000000 103:0.029703 104:0.000000 105:0.000000 106:0.019802

difficultés et limites

pour la réalisation de ce travail bien qu'il (projet) nous a été donné . Nous avons eu quelques difficultés lors des prérequis avec certaine paquets et module de python-opencv car nous avons choisi comme langage de programmation PYTHON et que nous avons pas pu finir ce projet.

conclusion :

de tout ce qui précédé, ce projet sur les descripteur global et local nous a permit de se familiariser avec le différente concept d'apprentissage automatique vu en cours et d'avoir une comprehension plus large sur l'indexation de contenu numerique

Annexes :

code du programme de Knn

```
# Utilisation
 les_K_Images_les_plus_similaires

# importation des packages necessaires
from scipy.spatial import distance as dist
import matplotlib.pyplot as plt
import numpy as np
import argparse
import glob
```

```
import cv2


# Construction du parseur d'arguments et parser les arguments
ap = argparse.ArgumentParser()
ap.add_argument("-f", "--nomfichier", required = True,
	help = "Nom du fichier passe en argument")
ap.add_argument("-M", "--valeurM", required = True,
	help = "Valeur pour la reduction de l'histogramme")
ap.add_argument("-N", "--valeurN", required = True,
	help = "nombre d'images ayant les plus petites distances par rapport a notre image requete")
'''ap.add_argument("-C", "--valeurC", required = True,
	help = "nombre de clusters pour K-MEANS")'''
args = vars(ap.parse_args())

print "\n\tImage de requete = " + args["nomfichier"]
print "\tValeur de M = " + args["valeurM"]
print "\tValeur de N = " + args["valeurN"]
#print "\tValeur de C = " + args["valeurC"]

# Initialise le dictionnaire d'index pour stocker le nom de l'image
# et les histogrammes correspondants et le dictionnaire d'images
# pour stocker les images elles-memes
index = {}
images = {}

# Boucle dans le dossier des images pour les recuperer et traiter
for imagePath in glob.glob("coil-100" + "/*.png"):
	# extraire le nom du fichier d’image
	# charger l’image, mettre à jour le dictionnaire des images
	filename = imagePath[imagePath.rfind("/") + 1:]
	image = cv2.imread(imagePath)
	images[filename] = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)

	# extraire un histogramme de couleur de l’image
	# using M bins per channel, normalize, and update Utiliser M binaire par cannal, normaliser
	# the index
```

```
        hist = cv2.calcHist([image], [0, 1, 2], None, [int(args["valeurM"]),
int(args["valeurM"]), int(args["valeurM"])],
                [0, 256, 0, 256, 0, 256])
        hist = cv2.normalize(hist).flatten()
        index[filename] = hist


        # Initialisation du dictionnaire
results = {}

print "\n\nI)KNN : K plus proches voisins(K images les plus similaires)\n"
print "\nImages"+ "\t\t\t" +"Distances\n"

        # Boucle sur l'index
for (k, hist) in index.items():
                # compute the distance between the two histograms
                # using the method and update the results dictionary
      d = dist.cityblock(index[args["nomfichier"]], hist)
      results[k] = d

        # sort the results
results = sorted([(v, k) for (k, v) in results.items()])


'''
   calcul des k plus proches voisins et deduction de la classe a laquelle appartient
l'image requete
'''
N=0
vote={}

for i in results :
   if N == int(args["valeurN"])+1 :
      break
   if i[0] != 0.0 :
      image_requete=cv2.imread("coil-100/"+args["nomfichier"])
      cv2.imshow("image requete",image_requete)
      voisin=cv2.imread("coil-100/"+ str(i[1]))
      cv2.imshow("voisin"+str(N),voisin)
      print "\""+str(i[1])+ "\"" + "\t\t" + str(i[0])
```

```
        classe=str(str(i[1]).split("_")[0])
        if classe in vote.keys() :
            vote[classe]=vote[classe]+1
        else :
            vote[classe]=1
    N=N+1


vote = sorted([(v, k) for (k, v) in vote.items()])
print "\nDetermination de la classe de l'image requete (classe majoritaire des ): " +
args["valeurN"] + " plus proches voisins"
print "\t"+str(vote)
print "La classe a laquelle appartient l'image est : " + vote[-1][1]
cv2.waitKey()
cv2.destroyAllWindows()
```

---

code programme moment de HU

```
# Utilisation
# python compare.py -f nomfichier -M valeurPourLareductionDeHistogramme -N
les_K_Images_les_plus_Similaires

# importation des packages necessaires
from scipy.spatial import distance as dist
import matplotlib.pyplot as plt
import numpy as np
import argparse
import glob
import cv2
from numpy import mgrid, sum
from math import sqrt

# Construction du parseur d'arguments et parser les arguments
ap = argparse.ArgumentParser()
ap.add_argument("-f1", "--nomfichier", required = True,
        help = "Nom du premier fichier passe en argument")
ap.add_argument("-f2", "--nomfichier2", required = True,
        help = "Nom du deuxieme fichier passe en argument")
```

```
ap.add_argument("-T", "--valeur", required = True,
        help = "Reduction du niveau de gris")
args = vars(ap.parse_args())

image=cv2.imread(args["nomfichier"])
image2=cv2.imread(args["nomfichier2"])
cv2.imshow("imageCouleur1",image)
cv2.imshow("imageCouleur2",image2)
image=cv2.cvtColor(image,cv2.COLOR_BGR2GRAY)
image2=cv2.cvtColor(image2,cv2.COLOR_BGR2GRAY)
cv2.imshow("imageGris1",image)
cv2.imshow("imageGris2",image2)

height1, width1 =image.shape[:2]
height2, width2 =image2.shape[:2]

for w in range(width1) :
   for h in range(height1) :
      image[w][h]=image[w][h]/int(args["valeur"])
      h=h+1
   w=w+1

for w in range(width2) :
   for h in range(height2) :
      image2[w][h]=image[w][h]/int(args["valeur"])
      h=h+1
   w=w+1

cv2.imshow("imageGrisReduction1",image)
cv2.imshow("imageGrisReduction2",image2)

def moments_func(image):

  assert len(image.shape) == 2 # only for grayscale images
  x, y = mgrid[:image.shape[0],:image.shape[1]]
  moments = {}
  moments['mean_x'] = sum(x*image)/sum(image)
  moments['mean_y'] = sum(y*image)/sum(image)

  # raw or spatial moments
```

```
  moments['m00'] = sum(image)
  moments['m01'] = sum(x*image)
  moments['m10'] = sum(y*image)
  moments['m11'] = sum(y*x*image)
  moments['m02'] = sum(x**2*image)
  moments['m20'] = sum(y**2*image)
  moments['m12'] = sum(x*y**2*image)
  moments['m21'] = sum(x**2*y*image)
  moments['m03'] = sum(x**3*image)
  moments['m30'] = sum(y**3*image)


  moments['mu11'] = sum((x-moments['mean_x'])*(y-moments['mean_y'])*image)
  moments['mu02'] = sum((y-moments['mean_y'])**2*image) # variance
  moments['mu20'] = sum((x-moments['mean_x'])**2*image) # variance
  moments['mu12'] = sum((x-moments['mean_x'])*(y-moments['mean_y'])**2*image)
  moments['mu21'] = sum((x-moments['mean_x'])**2*(y-moments['mean_y'])*image)
  moments['mu03'] = sum((y-moments['mean_y'])**3*image)
  moments['mu30'] = sum((x-moments['mean_x'])**3*image)


  moments['nu11'] = moments['mu11'] / sum(image)**(2/2+1)
  moments['nu12'] = moments['mu12'] / sum(image)**(3/2+1)
  moments['nu21'] = moments['mu21'] / sum(image)**(3/2+1)
  moments['nu20'] = moments['mu20'] / sum(image)**(2/2+1)
  moments['nu03'] = moments['mu03'] / sum(image)**(3/2+1)
  moments['nu30'] = moments['mu30'] / sum(image)**(3/2+1)
  moments['nu02'] = moments['mu02'] / sum(image)**(2/2+1)

  return moments

#Calcul des moments pour la premiere image
moments=moments_func(image)

HMI1_1=moments['nu02'] + moments['nu20']

HMI1_2= pow((moments['nu20'] - moments['nu02']),2) + 4*pow(moments['nu11'],2)

HMI1_3=pow((moments['nu30']-3*moments['nu12']),2)+ pow((moments['nu21']-
moments['nu03']),2)
```

HMI1_4=pow((moments['nu30']+moments['nu12']),2)+ pow((moments['nu21']+moments['nu03']),2)

HMI1_5=(moments['nu30']-3*moments['nu12'])*(moments['nu30']+moments['nu12'])*((pow(moments['nu30']+moments['nu12'],2))-3*(pow((moments['nu21']+moments['nu03']),2)))+(3*moments['nu21']-moments['nu03'])*(moments['nu21']+moments['nu03'])*(3*(pow(moments['nu30']+moments['nu12'],2))-(pow(moments['nu21']+moments['nu03'],2)))

HMI1_6=(moments['nu20']-moments['nu02'])*(pow(moments['nu30']+moments['nu12'],2)-(pow(moments['nu21']+ moments['nu03'],2)))+4*moments['nu11']*(moments['nu30']+moments['nu12'])*(moments['nu21']+moments['nu03'])

HMI1_7=(3*moments['nu21']-(moments['nu03']))*(moments['nu30']+moments['nu12'])*(pow(moments['nu30'],2)-3*(pow(moments['nu21']+moments['nu03'],2)))-(moments['nu30']-3*(moments['nu12']))*(moments['nu12']-moments['nu03'])*(3*(pow(moments['nu30']+moments['nu12'],2))-pow(moments['nu21']+moments['nu03'],2))

#Calcul des moments pour la deuxieme image

moments2=moments_func(image2)
HMI2_1=moments2['nu02'] + moments2['nu20']

HMI2_2= pow((moments2['nu20'] - moments2['nu02']),2) + 4*pow(moments2['nu11'],2)

HMI2_3=pow((moments2['nu30']-3*moments2['nu12']),2)+ pow((moments2['nu21']-moments2['nu03']),2)

HMI2_4=pow((moments2['nu30']+moments2['nu12']),2)+ pow((moments2['nu21']+moments2['nu03']),2)

HMI2_5=(moments2['nu30']-3*moments2['nu12'])*(moments2['nu30']+moments2['nu12'])*((pow(moments2['nu30']+moments2['nu12'],2))-3*(pow((moments2['nu21']+moments2['nu03']),2)))+(3*moments2['nu21']-moments2['nu03'])*(moments2['nu21']

```
+moments2['nu03'])*(3*(pow(moments2['nu30']+moments2['nu12'],2))-
(pow(moments2['nu21']+moments2['nu03'],2)))

HMI2_6=(moments2['nu20']-moments2['nu02'])*(pow(moments2['nu30']
+moments2['nu12'],2)-(pow(moments2['nu21']+ moments2['nu03'],2)))
+4*moments2['nu11']*(moments2['nu30']+moments2['nu12'])*(moments2['nu21']
+moments2['nu03'])

HMI2_7=(3*moments2['nu21']-(moments2['nu03']))*(moments2['nu30']
+moments2['nu12'])*(pow(moments2['nu30'],2)-3*(pow(moments2['nu21']
+moments2['nu03'],2)))-(moments2['nu30']-
3*(moments2['nu12']))*(moments2['nu12']-
moments2['nu03'])*(3*(pow(moments2['nu30']+moments2['nu12'],2))-
pow(moments2['nu21']+moments2['nu03'],2))

distance=sqrt((HMI1_1-HMI2_1)**2 + (HMI1_2-HMI2_2)**2 + (HMI1_3-
HMI2_3)**2 + (HMI1_4-HMI2_4)**2 + (HMI1_5-HMI2_5)**2 + (HMI1_6-
HMI2_6)**2 + (HMI1_7-HMI2_7)**2 )/7

print distance

cv2.waitKey()
```

---

code descripteur global

```
# Utilisation

# importation des packages necessaires
from scipy.spatial import distance as dist
import matplotlib.pyplot as plt
import numpy as np
import argparse
import glob
import cv2
from numpy import mgrid, sum
from math import sqrt


ap = argparse.ArgumentParser()
ap.add_argument("-f", "--nomfichier", required = True,
        help = "Nom du fichier passe en argument")
ap.add_argument("-M", "--valeurM", required = True,
        help = "Valeur pour la reduction de l'histogramme")
```

```
ap.add_argument("-N", "--valeurN", required = True,
        help = "nombre d'images ayant les plus petites distances par rapport a notre image
requete")
'''ap.add_argument("-C", "--valeurC", required = True,
        help = "nombre de clusters pour K-MEANS")'''
ap.add_argument("-poids", "--poids", required = True,
        help = "poids pour distance globale")
args = vars(ap.parse_args())

def prepareImage(image) :
   image=cv2.cvtColor(image,cv2.COLOR_BGR2GRAY)
   height, width =image.shape[:2]

   for w in range(width) :
      for h in range(height) :
         image[w][h]=image[w][h]/int(args["valeurM"])
         h=h+1
      w=w+1
   return image

def moments_func(image):

  assert len(image.shape) == 2 # only for grayscale images
  x, y = mgrid[:image.shape[0],:image.shape[1]]
  moments = {}
  moments['mean_x'] = sum(x*image)/sum(image)
  moments['mean_y'] = sum(y*image)/sum(image)

  # raw or spatial moments
  moments['m00'] = sum(image)
  moments['m01'] = sum(x*image)
  moments['m10'] = sum(y*image)
  moments['m11'] = sum(y*x*image)
  moments['m02'] = sum(x**2*image)
  moments['m20'] = sum(y**2*image)
  moments['m12'] = sum(x*y**2*image)
  moments['m21'] = sum(x**2*y*image)
  moments['m03'] = sum(x**3*image)
  moments['m30'] = sum(y**3*image)




  moments['mu11'] = sum((x-moments['mean_x'])*(y-moments['mean_y'])*image)
  moments['mu02'] = sum((y-moments['mean_y'])**2*image) # variance
  moments['mu20'] = sum((x-moments['mean_x'])**2*image) # variance
  moments['mu12'] = sum((x-moments['mean_x'])*(y-moments['mean_y'])**2*image)
  moments['mu21'] = sum((x-moments['mean_x'])**2*(y-moments['mean_y'])*image)
  moments['mu03'] = sum((y-moments['mean_y'])**3*image)
  moments['mu30'] = sum((x-moments['mean_x'])**3*image)
```

```
  moments['nu11'] = moments['mu11'] / sum(image)**(2/2+1)
  moments['nu12'] = moments['mu12'] / sum(image)**(3/2+1)
  moments['nu21'] = moments['mu21'] / sum(image)**(3/2+1)
  moments['nu20'] = moments['mu20'] / sum(image)**(2/2+1)
  moments['nu03'] = moments['mu03'] / sum(image)**(3/2+1)
  moments['nu30'] = moments['mu30'] / sum(image)**(3/2+1)
  moments['nu02'] = moments['mu02'] / sum(image)**(2/2+1)

  return moments



def moments_de_Hu(image) :

    moments=moments_func(image)
    HMI_1=moments['nu02'] + moments['nu20']

    HMI_2= pow((  moments['nu20'] -   moments['nu02']),2) + 4*pow(  moments['nu11'],2)

    HMI_3=pow((  moments['nu30']-3*  moments['nu12']),2)+ pow((  moments['nu21']-
moments['nu03']),2)

    HMI_4=pow((  moments['nu30']+  moments['nu12']),2)+ pow((  moments['nu21']+
moments['nu03']),2)

    HMI_5=(  moments['nu30']-3*  moments['nu12'])*(  moments['nu30']+
moments['nu12'])*((pow(  moments['nu30']+  moments['nu12'],2))-3*(pow((  moments['nu21']
+  moments['nu03']),2)))+(3*  moments['nu21']-  moments['nu03'])*(  moments['nu21']+
moments['nu03'])*(3*(pow(  moments['nu30']+  moments['nu12'],2))-(pow(  moments['nu21']
+  moments['nu03'],2)))

    HMI_6=(  moments['nu20']-  moments['nu02'])*(pow(  moments['nu30']+
moments['nu12'],2)-(pow(  moments['nu21']+   moments['nu03'],2)))+4*
moments['nu11']*(  moments['nu30']+  moments['nu12'])*(  moments['nu21']+
moments['nu03'])

    HMI_7=(3*  moments['nu21']-(  moments['nu03']))*(  moments['nu30']+
moments['nu12'])*(pow(  moments['nu30'],2)-3*(pow(  moments['nu21']+
moments['nu03'],2)))-(  moments['nu30']-3*(  moments['nu12']))*(  moments['nu12']-
moments['nu03'])*(3*(pow(  moments['nu30']+  moments['nu12'],2))-pow(  moments['nu21']+
moments['nu03'],2))

    return HMI_1, HMI_2, HMI_3, HMI_4, HMI_5, HMI_6, HMI_7


def distance_moments_de_Hu(moments_Hu_image1, moments_Hu_image2) :
```

```
    distance=sqrt((moments_Hu_image1[0]-moments_Hu_image2[0])**2 +
(moments_Hu_image1[1]-moments_Hu_image2[1])**2 + (moments_Hu_image1[2]-
moments_Hu_image2[2])**2 + (moments_Hu_image1[3]-moments_Hu_image2[3])**2 +
(moments_Hu_image1[4]-moments_Hu_image2[4])**2 + (moments_Hu_image1[5]-
moments_Hu_image2[5])**2 + (moments_Hu_image1[6]-moments_Hu_image2[6])**2 )/7

    return distance

def distance_globale(distance_descripteur_couleur,distance_moments_de_Hu, poids) :
    distance=poids*distance_descripteur_couleur + (1-poids)*distance_moments_de_Hu
    return distance

# Construction du parseur d'arguments et parser les arguments
print "\n\tImage de requete = " + args["nomfichier"]
print "\tValeur de M = " + args["valeurM"]
print "\tValeur de N = " + args["valeurN"]
#print "\tValeur de C = " + args["valeurC"]

# Initialise le dictionnaire d'index pour stocker le nom de l'image
# et les histogrammes correspondants et le dictionnaire d'images
# pour stocker les images elles-memes
index = {}
images = {}

# Boucle dans le dossier des images pour les recuperer et traiter
for imagePath in glob.glob("base" + "/*.png"):
        # extract the image filename (assumed to be unique) and
        # load the image, updating the images dictionary
        filename = imagePath[imagePath.rfind("/") + 1:]
        image = cv2.imread(imagePath)
        images[filename] = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)

        # extract a 3D RGB color histogram from the image,
        # using M bins per channel, normalize, and update
        # the index
        hist = cv2.calcHist([image], [0, 1, 2], None, [int(args["valeurM"]),
int(args["valeurM"]), int(args["valeurM"])],
                [0, 256, 0, 256, 0, 256])
        hist = cv2.normalize(hist).flatten()
        index[filename] = hist


        # Initialisation du dictionnaire
results = {}

print "\n\nI)KNN : K plus proches voisins(K images les plus similaires)\n"
print "\nImages"+ "\t\t\t" +"Distances\n"
```

```
moments_de_Hu_image_requete=moments_de_Hu(prepareImage(images[args["nomfichier"]
]))


        # Boucle sur l'index
for (k, hist) in index.items():
                # compute the distance between the two histograms
                # using the method and update the results dictionary
    d = dist.cityblock(index[args["nomfichier"]], hist)
    moments_de_Hu_image_courante=moments_de_Hu(prepareImage(images[k]))
    distance_moments_Hu=distance_moments_de_Hu(moments_de_Hu_image_requete,
moments_de_Hu_image_courante)
    results[k] = distance_globale(d,distance_moments_Hu, float(args["poids"]))


        # sort the results
results = sorted([(v, k) for (k, v) in results.items()])
```

---

code calcul des descripteur local pour chaque point d’intérêt

```
from os.path import exists, isdir, basename, join, splitext
import sift
from glob import glob
from numpy import zeros, resize, sqrt, histogram, hstack, vstack, savetxt, zeros_like
import scipy.cluster.vq as vq
import libsvm
from cPickle import dump, HIGHEST_PROTOCOL
import argparse


EXTENSIONS = [".jpg", ".bmp", ".png", ".pgm", ".tif", ".tiff"]
DATASETPATH = '../dataset'
PRE_ALLOCATION_BUFFER = 1000  # for sift
HISTOGRAMS_FILE = 'trainingdata.svm'
K_THRESH = 1  # early stopping threshold for kmeans originally at 1e-5, increased for
speedup
CODEBOOK_FILE = 'codebook.file'


def parse_arguments():
   parser = argparse.ArgumentParser(description='train a visual bag of words model')
   parser.add_argument('-d', help='path to the dataset', required=False,
default=DATASETPATH)
   args = parser.parse_args()
   return args


def get_categories(datasetpath):
```

```
    cat_paths = [files
                 for files in glob(datasetpath + "/*")
                  if isdir(files)]
    cat_paths.sort()
    cats = [basename(cat_path) for cat_path in cat_paths]
    return cats


def get_imgfiles(path):
    all_files = []
    all_files.extend([join(path, basename(fname))
                      for fname in glob(path + "/*")
                      if splitext(fname)[-1].lower() in EXTENSIONS])
    return all_files


def extractSift(input_files):
    print "extraction de caracteristiques SIFT"
    all_features_dict = {}
    for i, fname in enumerate(input_files):
        features_fname = fname + '.sift'
        if exists(features_fname) == False:
            print "calcul des caracteristiques SIFT pour ", fname
            sift.process_image(fname, features_fname)
        print "regroupement des caracteristiques SIFT pour ", fname,
        locs, descriptors = sift.read_features_from_file(features_fname)
        print descriptors.shape
        all_features_dict[fname] = descriptors
    return all_features_dict


def dict2numpy(dict):
    nkeys = len(dict)
    array = zeros((nkeys * PRE_ALLOCATION_BUFFER, 128))
    pivot = 0
    for key in dict.keys():
        value = dict[key]
        nelements = value.shape[0]
        while pivot + nelements > array.shape[0]:
            padding = zeros_like(array)
            array = vstack((array, padding))
        array[pivot:pivot + nelements] = value
        pivot += nelements
    array = resize(array, (pivot, 128))
    return array


def computeHistograms(codebook, descriptors):
    code, dist = vq.vq(descriptors, codebook)
    histogram_of_words, bin_edges = histogram(code,
```

```
                                    bins=range(codebook.shape[0] + 1),
                                    normed=True)
    return histogram_of_words


def writeHistogramsToFile(nwords, labels, fnames, all_word_histgrams, features_fname):
    data_rows = zeros(nwords + 1)  # +1 for the category label
    for fname in fnames:
        histogram = all_word_histgrams[fname]
        if (histogram.shape[0] != nwords):  # scipy deletes empty clusters
            nwords = histogram.shape[0]
            data_rows = zeros(nwords + 1)
            print ' n groupe a ete reduit a' + str(nwords)
        data_row = hstack((labels[fname], histogram))
        data_rows = vstack((data_rows, data_row))
    data_rows = data_rows[1:]
    fmt = '%i '
    for i in range(nwords):
        fmt = fmt + str(i) + ':%f '
    savetxt(features_fname, data_rows, fmt)



if __name__ == '__main__':
    print "----------------------"
    print "## chargement des images et extraction des caracteristiques "
    args = parse_arguments()
    datasetpath = args.d
    cats = get_categories(datasetpath)
    ncats = len(cats)
    print "chercher les dossiers de " + datasetpath
    if ncats < 1:
        raise ValueError('Only ' + str(ncats) + ' categories trouvees. chemin introuvable?')
    print "trouver les repertoire suivant / categorie:"
    print cats
    print "----------------------"
    all_files = []
    all_files_labels = {}
    all_features = {}
    cat_label = {}
    for cat, label in zip(cats, range(ncats)):
        cat_path = join(datasetpath, cat)
        cat_files = get_imgfiles(cat_path)
        cat_features = extractSift(cat_files)
        all_files = all_files + cat_files
        all_features.update(cat_features)
        cat_label[cat] = label
        for i in cat_files:
            all_files_labels[i] = label
```

```
    print "---------------------"
    print "## calcul des mots visuels par la methode K-means"
    all_features_array = dict2numpy(all_features)
    nfeatures = all_features_array.shape[0]
    nclusters = int(sqrt(nfeatures))
    codebook, distortion = vq.kmeans(all_features_array,
                                     nclusters,
                                     thresh=K_THRESH)

    with open(datasetpath + CODEBOOK_FILE, 'wb') as f:

        dump(codebook, f, protocol=HIGHEST_PROTOCOL)

    print "---------------------"
    print "## calcul des histogramme des mots visuels pour chaque image"
    all_word_histgrams = {}
    for imagefname in all_features:
        word_histgram = computeHistograms(codebook, all_features[imagefname])
        all_word_histgrams[imagefname] = word_histgram

    print "---------------------"
    print "## ecrire les histogramme"
    writeHistogramsToFile(nclusters,
                          all_files_labels,
                          all_files,
                          all_word_histgrams,
                          datasetpath + HISTOGRAMS_FILE)
```

---

code de classification des images

```
from os.path import exists, isdir, basename, join, splitext
import os
#import sift
from glob import glob
from numpy import zeros, resize, sqrt, histogram, hstack, vstack, savetxt, zeros_like
import scipy.cluster.vq as vq
#import libsvm
from cPickle import dump, HIGHEST_PROTOCOL
import argparse




def get_categories(datasetpath):
    cat_paths = [files
                 for files in glob(datasetpath + "/*")
                ]
    cat_paths.sort()
```

```
    for cat_path in cat_paths :
        base=basename(cat_path).split('_')[0]
        if not exists(base) and not isdir(base) :
            os.system("mkdir " + base)
        os.system("mv " + cat_path + " " +  base)



if __name__ == '__main__':
    get_categories("coil-100")
```