Indexation des Contenus Multimédia

AUTEUR:KAFANDO RODRIQUE & KOUADIO KOUAME OLIVIER

Projet Final

SOMMAIRE

AUTEUR:KAFANDO RODRIQUE & KOUADIO KOUAME OLIVIER

Projet Final

INTRODUCTION

L'évolution de la technologie à touché plusieurs secteurs, d'où l'apparition d'appareils d'acquisition d'images qui produisent un nombre important d'images chaque année. Cependant l'accès rapide à ces bases d'images énormes nécessite des algorithmes d'indexation efficaces. Ce rapport pressente le travail effectué dans le cadre de notre TP, il est composé de deux parties d'abord l'utilisation des descripteurs Globaux sur des images. Dans cette nous implémenterons un système de recherche d'images similaires(KNN),les descripteurs globaux du contenu d'images (histogrammes et moment de Hu), et un clustering. Dans la deuxième partie nous aborderons les descripteurs locaux.

Pour des raisons de temps de calcul nous avons réduire la base d'images a 1337 images au lieu de 7200, elle est composée de chaque type d'image.

I. Description globale du contenu des image

I.1- KNN avec les histogrammes couleurs

le calcul de l'histogramme couleur des images permet de trouver une image similaire à une image requête par la caractéristique couleur. Pour fonctionner le programme prend en paramètre une image requête (encadré en bleu sur les illustrations), le nombre de bits pour les différentes histogrammes nous avons utilisé dans ce programme 8, 16, et 32 bits puis le K. Le calcule de la distance s'effectue entre l'image requête et l'ensemble des images de la base, les résultats obtenus nous ont permis de constater la différence ou l'écart qui existe entre cette dernière par rapport aux autres en terme de ressemblance ou de similarité. Nous avons retenus **10 plus proches voisins** pour chaque nombre de bit histogramme choisi. Les images ci-dessous présentent les résultats obtenus.

```
Distance globale avec histogrammme a 8 bits
obj1__0.png = 0.0
obj1__10.png = 0.0634870499052
obj1__15.png = 0.100442198358
obj1__5.png = 0.132027795325
obj2__25.png = 0.456411876184
obj2__30.png = 0.536007580543
obj2__35.png = 0.642451042325
obj2__40.png = 0.687302590019
obj3__15.png = 0.835123183828
obj3__20.png = 0.851547694251
```

Illustration 1: distance histogramme à 8 bits

Illustration 2 distance histogramme à 16 bits

Illustration 3: distance histogramme à 32 bits

la première distance obtenue est celle de l'image requête cette valeur est 0.0 (encadré en bleu sur les illustrations) le programme, a partir de cette image va parcourir toute la base d'image et rechercher les images qui sont similaires à l'image passée en paramètre ensuite il affiche dans l'ordre croissant (par le tri) les distances entre l'image requête et les autres sur 8, 16 et 32 bits. Il faut noter que les images dont les distances sont proches de 0.0 sont des images qui sont similaire à l'image reqête donc celles dont les distances s'éloignent de 0.0 sont des images qui sont différentes de l'image requête.

Les résultats ci-dessus montrent également que lorsque le nombre de bits est petit plus la distance est grande vis-versa . Lorsque le nombre de bit de l'histogramme est petit la précision n'est pas très perceptible donc pour avoir une meilleure précision, il est nécessaire d'utiliser un histogramme à grande échelle cela

est bien perceptible sur la troisième illustration (histogramme à 32). Ce résultat donne des petites distances. Ces petites distance montre si l'image trouvée est similaire a l'image requête.
Cependant nous avons contacté que, plus le nombre de bit de l'histogramme est grand plus le temps de calcul est beaucoup important, cela est normal car comme nous venons de le souligner un histogramme à grande échelle donne une meilleur résultats. Nous remarquons qu'en dépit des changements effectués au niveau des échelles de l'histogramme nous observons que l'image qui est plus similaire à notre image requête reste inchangé: l'image obj1_10. png

I.2- KNN avec les moments de Hu

les moments de Hu permettent de calculer la distance entre une image cible et les autres images de la base, en se basant, contrairement des histogrammes couleurs sur la forme et affiche le résultat, Pour l'utiliser il faut rendre les images couleur en niveau de gris. Le programme calcul d'abord le moment de l'image requête, ensuite calcul le moment des N images contenues dans la base et affiche comme dans les étapes précédents les 10 plus proches voisins c'est à dire les images dont les distances tendent vers celle de l'image requête. l'image ci-après montre le résultat obtenu.

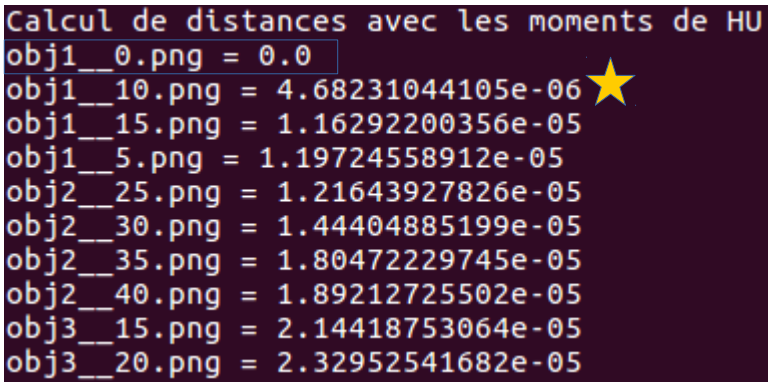


Illustration 4: distance avec les moments de Hu

Même remarque faite dans la partie précédente, l'image la plus similaire a notre image cible reste inchangé, l'image obj1__10.png est l'image qui possède la plus petite distance par rapport à notre image requête choisie. Par le tri effectué nous constatons que la position des images en terme de distance n'a pas changé quand bien même que les valeurs des distances obtenues varient en fonction de la méthode appliquée. Il faut noter que avec les moments de Hu nous obtenons des très petites distances entre notre image requête et les autres images de la base, cela signifie que avec les moments de Hu nous avons plus de précision sur la caractéristiques de similarité donc le programme détecte facilement les images similaire . Au vu de ce

qui précède nous nous dire aisément que la méthode de calcul de détection de similarité par les moments de Hu est plus adapté par rapport à la méthode d'histogramme couleur.
Le problème de rencontré ici est que les moments de Hu génèrent les plus petites distances mais dans la réalité il détecte des image qui sont pas de la même classe que l'image requête cela est dur au fait le moment de Hu se base sur plusieurs caractéristiques (plusieurs critères) pour claculer les distances par exemple la rotation, le centre, la couleur ...

I.3- Les distances globales

Dans cette partie nous combinons le calcul des distances de moment de Hu et celui des histogramme couleurs ceux en fonction des différents échelles des histogramme (8 16 32 bits). Les images ci-dessous présentes les résultats obtenus

```
Calcul de la distance globale(histogramme 8 bits + moment de hu)
obj1__0.png = 0.0
obj1__10.png = 0.0317507451969
obj1__15.png = 0.050223440334
obj1__5.png = 0.0660255452897
obj2__25.png = 0.228215398729
obj2__30.png = 0.26801665942
obj2__35.png = 0.321242546259
obj2__40.png = 0.343672764187
obj3__15.png = 0.417632386153
obj3__20.png = 0.425843047811
```

Figure 5 : distance globale (histogramme 8 bits+ moment de Hu)

```
Calcul de la distance globale(histogramme 16 bits + moment de hu)
obj1__0.png = 0.0
obj1__10.png = 0.0327175006181
obj1__15.png = 0.0556982239539
obj1__5.png = 0.064548146743
obj2__25.png = 0.205238053715
obj2__30.png = 0.246918653438
obj2__35.png = 0.286200404738
obj2__40.png = 0.31057969095
obj3__15.png = 0.457888424954
obj3__20.png = 0.461678328594
```

Figure 6: distance globale (histogramme 16 bits+ moment de Hu)

```
Calcul de la distance globale(histogramme 32 bits + moment de hu)
obj1__0.png = 0.0
obj1__10.png = 0.0170757104992
obj1__15.png = 0.0184377731608
obj1__5.png = 0.0330997831283
obj2__25.png = 0.171951047455
obj2__30.png = 0.193330097237
obj2__35.png = 0.202440954957
obj2__40.png = 0.209733474823
obj3__15.png = 0.455822210921
obj3__20.png = 0.457978745693
```

Figure7 : distance globale (histogramme 32 bits+ moment de Hu)

Depuis le début nous avons essayé de suivre le résultat de la distance entre l'image obj1__10.png par avec notre image requête(obj1__0.png) , en observant avec une grande attention nous constatons que le calcule des distances globales (histogrammes + moment de Hu) réduit considérablement les distance entre l'image requête et les autres images. Il donne un résultat optimisé. Avec ces résultats obtenus, nous affirmons que plus le nombre de bits a considérer dans l'histogramme est élevé plus il y a un meilleur résultat par exemple le cas d'histogramme 32 bits . En effet un histogramme avec une grand nombre de bits donne plus d'information sur l'image ce qui facilite la recherche dans la base car les caractéristiques de similarité sont apparaissent bien en d'autre terme le programme détecte facilement ces points de similarité puisqu'il a beaucoup d'information sur l'image.
Il faut note lorsque le distance du moment de Hu est concaténé avec la distance d'histogramme couleur cela donne un résultat optimisé par rapport à la distance d'histogramme simple .

I.4- Comparaison avec des résultats des descripteurs

Dans cette partie de notre rapport nous faisons une analyse comparative des résultats obtenus. Pour que nous soyons claire nous allons reporter nos résultats dans un tableau et faire analyse. Cette tableau-ci après montre les 05 images les plus similaires de la base d'image à l'image requête.

Nom Image	Distance Histogramme		Moment de HU	Distance Globale	
	8 bits	32 bits		8 bits	32 bits
Nom image requête: obj1__0.png: 0.0					
Obj1__10.png	0.0634870499052	0.0341467386879	4.68231044105e-06	0.03117507451969	0.0170757104992
Obj1__15.png	0.100442198358	0.0368611058332	1.162922000356e-05	0.050223440334	0.0184377731608
Obj1__5.png	0.132027795325	0.0661762710024	1.19724558912e-05	0.0660255452897	0.0330997831283
Obj1__25.png	0.456411876184	0.343883173638	1.21643927826e-05	0.228215398729	0.171951047455
Obj1__30.png	0.536007580543	0.386634456177	1.44404885199e-05	0.26801665942	0.193330097237

Tableau 1: comparaison des differents technique de détection d'image avec 5 NN

A l'analyse du tableau nous observons que les résultats optimisé ici sont ceux des moments de Hu il donne des distances très réduit entre l'image requête et les autres images pour détecter les plus similaires en fonction du nombre de 5 voisins plus proches entrée . En oppossant les données nous constatons dans le calcul des distances globales (Histogramme+moments de Hu) donne des résultats plus réduit par rapport au distance Histogramme cela est dû simplement au fait que les données des moments de Hu influencent considérablement le résultat des calculs avec les histogrammes couleurs. En analysant le tableau des résultats obtenus, nous pouvons conclure que la methode des moments de Hu est plus efficace et est plus adaptée par rapport à la methode des histogramme couleurs .

Illustration 7: obj1__0.png

Illustration 6: obj1__10.png

Illustration 5: obj1__30.png

IMAGE REQUÊTE

IMAGE AVEC PLUS PETITE DISTANCE (5NN)

IMAGE AVEC PLUS GRANDE DISTANCE (5NN)

Nous observons que dans la réalité le modeler détecte bien les images similaires a l'image requête

II. Clustering analyse des expérimentations

Dans cette partie, nous avons effectué une classification des différentes images en les affectant à des centres que nous avons préalablement défini.

Principe

A partir des centres défini, On affecte aléatoirement chaque image à l'un des centres et on itère comme suit: les centres des différents groupes sont recalculés et chaque image est de nouveau affecté à un centre en fonction du centre le plus proche c-à-dire le centre avec le quel la distance est moins élevée. La convergence est atteinte lorsque les centres sont fixes.

Phase d'expérimentation

Afin de pouvoir évaluer notre programme, nous avons effectuer une classification de la base d'image en utilisant la distance entre les histogramme. Pour des raison de temps de calcul, nous avons fixer le nombre de bits à 64 bits.
Nous avons tester avec différentes valeurs de k et les résultats illustrés ci-dessous sont ceux pour k=15.

Pour k=15

Centroides	classe	Nombre d'images
'obj2__55.png'	0	329 (35 classes d'images)
'obj3__65.png'	1	36 (2 classes d'images)
'obj4__5.png'	2	36 (2 classes d'images)
'obj5__5.png'	3	84 (9 classes d'images)
'obj7__25.png'	4	293 (23 classes d'images)
'obj6__105.png'	5	513 (32 classes d'images)
'obj8__10.png'	6	303 (……..)
'obj9__40.png'	7	863 (……..)
'obj10__200.png'	8	63 (……..)
'obj11__55.png'	9	245 (……..)
'obj12__20.png'	10	39 (……..)
'obj13__10.png'	11	147 (……..)
'obj14__100.png'	12	52 (……..)
'obj16__20.png'	13	126 (……..)
'obj2__55.png'	14	270 (……..)

A l'issu des tests que nous avons effectué et de ces différents résultats, nous avons pu remarquer que plus la valeur de k est élevée, plus nous avons une bonne classification. Par exemple, si nous prenons la première classe, nous avons obtenu une classification de 329 d'images autours du centroïde 'obj2__55.png' soit une composition de 35 images différentes. Plusieurs raisons expliquent les erreurs de classification. Il s'agit essentiellement des erreurs de calcul liées à la réduction du nombre bits (de 256 à 64) pour l'échantillonnage et à la valeur de K fixé. Plus la valeur de K est grande, plus on a un bon taux de classement.

DEUXIÈME PARTIE : Descripteurs locaux et approche Bag-of-words.

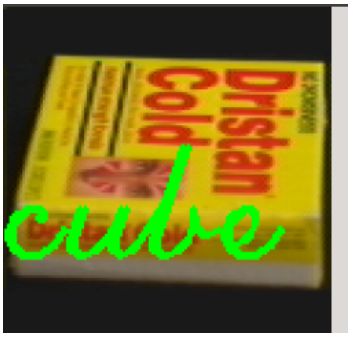
Le programme est composé 3 fichiers : un fichier findfeature.py qui permet de récupérer les caractéristiques de l'image , un fichier getclass.py qui lui s'occupera de récupérer la classe à laquelle l'image appartient et un fichier imutils.py qui permet de lire et afficher les images.

I. Phase d'apprentissage

Dans cette partie nous avons crée une base d'apprentissage avec les différentes types d'image ce qui permettra a notre programme d'apprendre les images pour ensuite les reconnaître en fonction des caractéristiques , les distance prise en compte sont celle des distances d'histogramme couleurs . Ensuite nous avons appliqué la methode de SVM pour faire notre apprentissage . Il est important pour nous de donner le fonctionnement des la méthode SVM que nous avons utilisé: nous disons que SVM fonctionne par mappage des données à un espace d'attributs haute dimension pour que les points de données puissent être classés, même lorsque les données ne sont pas séparables sur un plan linéaire. Un séparateur entre les catégories est identifié. Ensuite, les données sont transformées de sorte que le séparateur puisse être défini comme un hyperplan. Ensuite, les caractéristiques des nouvelles données peuvent être utilisées pour prédire le groupe auquel un nouvel enregistrement doit appartenir. Nous avons utilisé le SVM de Opencv pour appliquer sur les résultats du modèle K_means préalablement obtenue.

II. Phase de test

Cette phase de test nous a permit d'évaluer l'efficacité de notre modèle , elle permettra de savoir si notre modele a bien appris et s'il pourra faire le bon classement . Alors pour cela nous lançons trois classes (cube, oignon et tomate) d'image. Puisque le modèle a fait l'apprentissage il devra nous classer les image image que nous lui passons en paramètres. Les images ci-dessus sont les résultats obtenus



Au regard des résultats obtenue nous pouvons dire que notre modèle a fait un bon apprentissage car il arrive a reconnaître les images que nous lui donnons en paramètre donc il fait un bon classement.

CONCLUSION

Cet TP nous a permis de calculer tout d'abord de comprendre le fonctionnement des descripteurs. En effet nous avons eu à calculer les distances entre une image requête et les autres image d'une base, en utilisant les techniques d'histogrammes couleurs , les moments de Hu et les distances globales. Après implémentation les résultats obtenus montrent que la technique des moments de Hu est la technique la plus efficace.

Dans la deuxième partie avec la technique des k-mens nous avons classifié les images et nous avons appliqué les SVM pour faire l'apprentissage à partir d'une base d'apprentissage conçue et le test sur la base d'images de test. Après Exécution nous le modèle donne un bon résultat .

Références

[1] : https://en.wikipedia.org/wiki/Image_moment
[2] :http://docs.opencv.org
[3] :www.google.com.vn

■ Annexe

✗ code source de la première partie descripteur.

```
###UTILISATION#####################################
#
# python Descripteurs.py 'nom_image_requette'
#################################################""


## importation des packages
import pickle
from scipy.spatial import distance as dist
import numpy as np
np.seterr(over='ignore')
import argparse
import glob
import cv2
from sklearn.cluster import KMeans
from matplotlib import pyplot as plt
import os
from math import sqrt
import csv
import random
import math
import operator

    #fonction de calcul d'histogramme
def histogramme(rep, img):

    nb_bits=8

    x=1
    hist=[]
    #lecture des images de la base
```

```
    img = cv2.imread(rep + img)
    #split de la base de couleur RGB
    b, g, r = cv2.split(img)
    # histogramme de B
    histrb = cv2.calcHist([img],[0],None,[nb_bits],[0,nb_bits])
    #histogramme de G
    histrg = cv2.calcHist([img],[1],None,[nb_bits],[0,nb_bits])
    #histogramme R
    histrr = cv2.calcHist([img],[2],None,[nb_bits],[0,nb_bits])
    #sommation des histogramme de RGB
    for i in range(len(histrb)):
        hist.append( float(histrb[i]))
    for i in range(len(histrg)):
        hist.append( float(histrg[i]))
    for i in range(len(histrr)):
        hist.append( float(histrr[i]))
    return hist # returne la valeur de l'histogramme des trois couleurs
# Calcul de la distancee entre deux images de la base
def distance(histA, histB):
    histg=0
    d_global=[] # liste des distances
    #parcour des valeurs des histogrammes A t B
    for i in range(len(histA)):

        histg = abs(float(histA[i])- float(histB[i]))
        d_global.append( histg/sum(histA))# insere toutes las valeurs dans la liste

    return     float(sum(d_global)) # retourne la distance entre les deux histogrammes

##Implementation manuelle du moment de Hu "reference wikippedia"
"""
def BasicMoment(p,q,img):
    mpq=0

    b=[]
    x, y, c= img.shape
    for i in range(x):
        for j in range(y):
                mpq +=(img[i,j]*(i**p)*(j**q))
```

```
    return mpq

def CentraMoment(p,q,img):
    Mpq=0
    img = cv2.imread('image_files/obj1__0.png')
    x, y, c = img.shape
    i0=BasicMoment(1,0,img)/BasicMoment(0,0,img)
    j0= BasicMoment(0,1,img)/BasicMoment(0,0,img)
    for i in range(x):
        for j in range(y):
                Mpq=Mpq+sum(img[i,j]*pow(i-i0,p)*pow(j-j0,q))
    return Mpq


def NormaMoment (p,q,img):
    U=0
    Mpq=CentraMoment(p,q,img)
    gama= (p+q/2) +1
    Moo=CentraMoment(0,0,img)
    U= Mpq/math.pow(Moo,gama)
    return U

def hmi (img):
        HMI=[]
        HMI1= NormaMoment(0,2,img) +NormaMoment(2,0,img)
        HMI2=                    (NormaMoment(2,0,img)-NormaMoment(0,2,img))**2
+4*NormaMoment(1,1,img)**2
        HMI3=                  (NormaMoment(3,0,img)-3*NormaMoment(1,2,img))**2
+(3*NormaMoment(2,1,img)-NormaMoment(0,3,img)**2)
        HMI4=          (NormaMoment(3,0,img)+NormaMoment(1,2,img))**2          +
(NormaMoment(2,1,img)+NormaMoment(0,3,img)**2)
        HMI5=                                                    (NormaMoment(3,0,img)-
3*NormaMoment(1,2,img))*(NormaMoment(3,0,img)
+NormaMoment(1,2,img))*((NormaMoment(3,0,img)+NormaMoment(1,2,img))**2
-3*                          (NormaMoment(2,1,img)+NormaMoment(0,3,img)**2))
+(3*NormaMoment(2,1,img)-NormaMoment(0,3,img)*NormaMoment(2,1,img)
+NormaMoment(0,3,img))*(3*(NormaMoment(3,0,img)
+NormaMoment(1,2,img))**2-(NormaMoment(2,1,img)
+NormaMoment(0,3,img)**2))
```

```
        HMI6=(NormaMoment(2,0,img)-NormaMoment(0,2,img))*
(((NormaMoment(3,0,img)+NormaMoment(1,2,img))**2)-
(NormaMoment(2,1,img)+NormaMoment(0,3,img))**2)                    +
4*NormaMoment(1,1,img)*(NormaMoment(3,0,img)+
NormaMoment(1,2,img))*(NormaMoment(2,1,img)+NormaMoment(0,3,img))
        HMI7=          (3*NormaMoment(2,1,img)-          NormaMoment(0,3,img))*
(NormaMoment(3,0,img)+    NormaMoment(1,2,img))*    ((NormaMoment(3,0,img)
+NormaMoment(1,2,img))**2              -              3*(NormaMoment(2,1,img)+
NormaMoment(0,3,img))**2)               -               (NormaMoment(3,0,img)-
3*(NormaMoment(1,2,img)))*              (NormaMoment(1,2,img)              +
NormaMoment(0,3,img))*(3*(NormaMoment(3,0,img)+
NormaMoment(1,2,img))**2               -               (NormaMoment(2,1,img)+
NormaMoment(0,3,img))**2)
        HMI.append(HMI1)
        HMI.append(HMI2)
        HMI.append(HMI3)
        HMI.append(HMI4)
        HMI.append(HMI5)
        HMI.append(HMI6)
        HMI.append(HMI7)
        return HMI

def DistanceHu(img1, img2):
    dist=0
    moment1= hmi(img1)
    moment2= hmi(img2)

    for i in range (len(moment1)):
        dist+= (moment1[i]-moment2[i])**2
    DisFinal= np.sqrt(dist)/7

    return    DisFinal
"""


### Calcul du moment du Hu en utilisant la fonction de opencv
def MomentHU (img):
    #calul du moment de l'image
    HU=cv2.HuMoments(cv2.moments(img)).flatten()
```

```
    return HU # return le moment de hu de l'image


def DistanceHu(im1,im2):
    dist=0
    moment1= MomentHU(im1)# moment de l'image1
    moment2= MomentHU(im2)# moment de l'image2
    #parcourdes valeurs du moment
    for i in range (len(moment1)):
        #calcul de la distance
        dist= dist +((moment1[i]-moment2[i])**2)
        DisFinal= np.sqrt(dist)/7
    return   DisFinal # retourne la distance de HU
#Calcul de la distance globale
def Global_fonction (histA,histB,img1,img2):
        D_glob= 0.5*distance(histA,histB)+ 0.5*DistanceHu(img1,img2)
        return D_glob # Retourne la distance globale



def main():
    # variables
    img1 = cv2.imread('image_files/obj1__0.png',0)

rep='/media/rdius/DONNEES/IFI/INDEXATION/TP1_INDEX/TP_INDEXATION/
TP_INDEX_PARTIE_1/images/'
    all_files = os.listdir(rep)
    nb_img=len(all_files)
    i=1
    histA=histogramme(rep, "obj1__0.png")
    #histB=histogramme(rep, "obj1__30.png")
    #d= distance(histA,histB)
    histB=[]
    d1=[]
    d2=[]
    d3=[]
    k=1
    N=[]
    neighbors = []
```

```
#img1 = cv2.imread('image_files/obj1__0.png',0)
#moment1= hmi(img1)
c=0
D=0
i=1
img=[]
#print img1
D_glob=0
#print "Distance globale avec histogrammme a 8 bits"
#print "Calcul de distances avec les moments de HU"
print "Calcul de la distance globale(histogramme 8 bits + moment de hu)"
tri1 =[];tri2 =[];tri3 =[]

while i <nb_img-1:
    histB=histogramme(rep, all_files[i])
    img2 = cv2.imread('images/' + all_files[i], 0)
    #fonction de calcul da la distance d'histogramme
    D_hist=distance(histA,histB)
    d1.append(D_hist)
    tri1=sorted(d1)
    #Fonction de calcul du moment de hu
    D_HU=DistanceHu(img1,img2)
    d2.append(D_HU)
    tri2=sorted(d2)
    ##fonction de calcul distance globale
    D_glob=Global_fonction(histA,histB,img1,img2)
    d3.append(D_glob)
    tri3=sorted(d3)
    i=i+1

    #Affichage des distace entr histogramme
    """
for i in range (len(tri1)):
    print all_files[i],"=", tri1[i]
    """
    ##Affichage des distance moments de hu
for i in range (len(tri2)):
    print all_files[i],"=", tri2[i]
    #Affichage distance globale
```

```
        """
    for i in range (len(tri3)):
        print all_files[i],"=", tri3[i]
    """
if __name__ == "__main__":
        main()
```

- code source deuxième partie: SIFT
- **fichier findFeatures.py**

```
#!/usr/local/bin/python3.6

import argparse as ap
import cv2
import imutils
import numpy as np
import os
from sklearn.svm import LinearSVC
from sklearn.externals import joblib
from scipy.cluster.vq import *
from sklearn.preprocessing import StandardScaler

# chemin d'aces de la base d'aprentissage
parser = ap.ArgumentParser()
parser.add_argument("-t", "--trainingSet", help="Path to Training Set",
required="True")
args = vars(parser.parse_args())

# Recceuil et sauvegarde les classes de la base d'apprentissage
train_path = args["trainingSet"]
training_names = os.listdir(train_path)

# Sauvegarder du chemin d'acces aux images et les étiquettes #correspondantes
image_paths = []
image_classes = []
class_id = 0
for training_name in training_names:
    dir = os.path.join(train_path, training_name)
    class_path = imutils.imlist(dir)
```

```
    image_paths+=class_path
    image_classes+=[class_id]*len(class_path)
    class_id+=1

# Construction des caracteristiques et des points de detection

feat_det = cv2.xfeatures2d.SIFT_create()

# Liste pour stocker les descripteurs
des_list = []

for image_path in image_paths:
    im = cv2.imread(image_path)
   # kpts = fea_det.detect(im)
    #kpts, des = des_ext.compute(im, kpts)
    kpts, des = feat_det.detectAndCompute(im,None)
    des_list.append((image_path, des))

# Stack all the descriptors vertically in a numpy array
descriptors = des_list[0][1]
for image_path, descriptor in des_list[1:]:
    descriptors = np.vstack((descriptors, descriptor))

# Classificateur k-means
k = 100
voc, variance = kmeans(descriptors, k, 1)

# Calcul de dsecripteurs (histogramme)
im_features = np.zeros((len(image_paths), k), "float32")
for i in range(len(image_paths)):
    words, distance = vq(des_list[i][1],voc)
    for w in words:
        im_features[i][w] += 1

# Vectorisation
nbr_occurences = np.sum( (im_features > 0) * 1, axis = 0)
idf = np.array(np.log((1.0*len(image_paths)+1) / (1.0*nbr_occurences + 1)), 'float32')

# Mise à l'echelle des caracteristiques des mots
```

```
stdSlr = StandardScaler().fit(im_features)
im_features = stdSlr.transform(im_features)

# Apprentissage à base du vecteur SVM
clf = LinearSVC()
clf.fit(im_features, np.array(image_classes))

# Sauvegarde des données
joblib.dump((clf, training_names, stdSlr, k, voc), "bof.pkl", compress=3)
```

✗ fichier getclass.py

```
#!/usr/local/bin/python3.6

import argparse as ap
import cv2
import imutils
import numpy as np
import os
from sklearn.svm import LinearSVC
from sklearn.externals import joblib
from scipy.cluster.vq import *

# Charge les classificateurs, les classes, les noms et le nombres de cluster et le vocabulaire associée
clf, classes_names, stdSlr, k, voc = joblib.load("bof.pkl")

# Acces au chemin de la base de test
parser = ap.ArgumentParser()
group = parser.add_mutually_exclusive_group(required=True)
group.add_argument("-t", "--testingSet", help="Path to testing Set")
group.add_argument("-i", "--image", help="Path to image")
parser.add_argument('-v',"--visualize", action='store_true')
args = vars(parser.parse_args())

# Acces et sauvegarde de la base de test
image_paths = []
if args["testingSet"]:
```

```
    test_path = args["testingSet"]
    try:
        testing_names = os.listdir(test_path)
    except OSError:
        print ("Fichier inconnu".format(test_path))
        exit()
    for testing_name in testing_names:
        dir = os.path.join(test_path, testing_name)
        class_path = imutils.imlist(dir)
        image_paths+=class_path
else:
    image_paths = [args["image"]]

# Création et extraction des caracteristiques des points de detection
feat_det = cv2.xfeatures2d.SIFT_create()

# Liste dans laquelle sera stockée l'ensemble des descripteurs
des_list = []

for image_path in image_paths:
    im = cv2.imread(image_path)
    if im == None:
        print ("Fichier inconnu".format(image_path))
        exit()
    kpts, des = feat_det.detectAndCompute(im,None)
    des_list.append((image_path, des))

# Empilage des descripteurs dans un tableu
descriptors = des_list[0][1]
for image_path, descriptor in des_list[0:]:
    descriptors = np.vstack((descriptors, descriptor))

#
test_features = np.zeros((len(image_paths), k), "float32")
for i in range(len(image_paths)):
    words, distance = vq(des_list[i][1],voc)
    for w in words:
        test_features[i][w] += 1
```

```
# Vectorisation
nbr_occurences = np.sum( (test_features > 0) * 1, axis = 0)
idf = np.array(np.log((1.0*len(image_paths)+1) / (1.0*nbr_occurences + 1)), 'float32')

# Echelle des fonctionnalite
test_features = stdSlr.transform(test_features)

# Prediction des classes
predictions =  [classes_names[i] for i in clf.predict(test_features)]

#Visualisation des résultats, si l'indicateur "visualiser" est défini #sur true par l'utilisateur
if args["visualize"]:
    for image_path, prediction in zip(image_paths, predictions):
        image = cv2.imread(image_path)
        cv2.namedWindow("Image", cv2.WINDOW_NORMAL)
        pt = (0, 3 * image.shape[0] // 4)
         cv2.putText(image, prediction, pt ,cv2.FONT_HERSHEY_SCRIPT_COMPLEX, 2, [0, 255, 0], 2)
        cv2.imshow("Image", image)
        cv2.waitKey(3000)
```

✗ fichier imutils.py

```
#!/usr/local/bin/python3.6

import cv2, os
import matplotlib.pyplot as plt

def imlist(path):
    """
       Cette fonction permet de renvoyer les noms des images dans le chemin du du repertoire fourni comme argument
    """
    return [os.path.join(path, f) for f in os.listdir(path)]

def imshow(im_title, im):
    ''' Affichage des images'''
```

```
    plt.figure()
    plt.title(im_title)
    plt.axis("off")
    if len(im.shape) == 2:
        plt.imshow(im, cmap = "blue")
    else:
        im_display = cv2.cvtColor(im, cv2.COLOR_RGB2BGR)
        plt.imshow(im_display)
    plt.show()

def imreads(path):
    '''
     Renvoit les resultats des images lu dans un dossier
    '''
    images_path = imlist("/home/bikz05/Desktop/back_projection")
    images = []
    for image_path in images_path:
        images.append(cv2.imread(image_path, cv2.CV_LOAD_IMAGE_COLOR))
    return images

def show(image, name="Image"):
    '''
    Routine d'affichage des images
    '''
    cv2.namedWindow(name, cv2.WINDOW_NORMAL)
    cv2.imshow(name, image)
    cv2.waitKey(0)
```