Rapport de TP1 Vision par Ordinateur

Rediger par : **OLEMBO Réel Devin Richmond**
Date: 13/11/2019

INTRODUCTION

Dans le cadre de notre cours de vision par ordinateur il nous a été demandé de faire la reconnaissance d'objets à l'aide du descripteur SIFT. Notre l'application prendra en entrée une image et donne en sortie le nom de l'objet contenu sur l'image. Nous allons tester et comparez notre programme sur deux bases d'images différentes pour comparer la précision de ces bases.

1. Présentation de l'application

Premièrement pour pouvoir utiliser notre application il faudrait s'assurer que tous les dépendances sont installées, pour se faire il nous faut installer les bibliothèques python: os, numpy, pickle, seaborn, pandas, matplotlib, opencv.
Ensuite pour utiliser l'application, il suffit juste lancer le fichier de détection en tapant la commande **python detect.py** en donnant en paramètre une image.
Un fichier **test_train_matrixConf.py** est prevu pour l'apprentissage

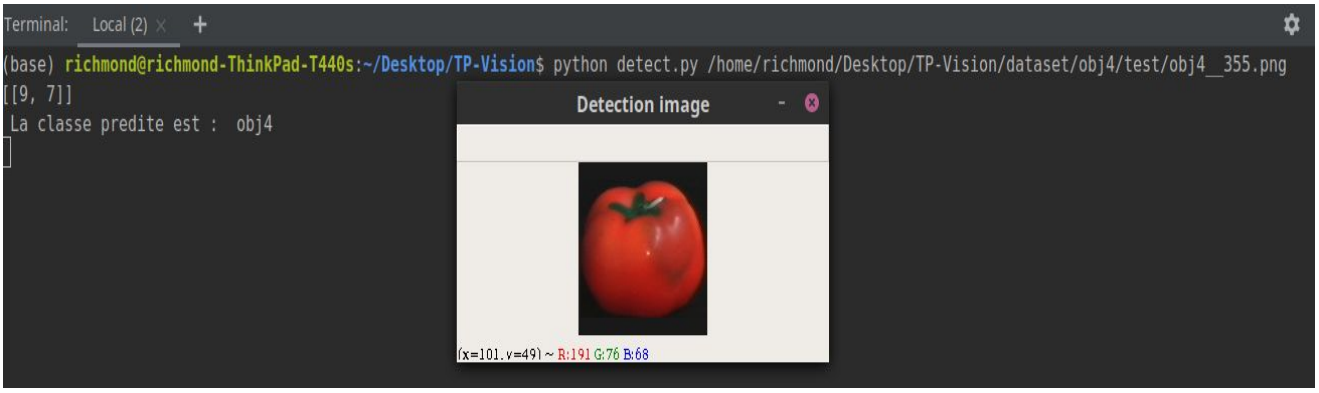

Figure 1: exemple d'une image détecter

2. Présentation du jeu de données

Pour notre travaille nous avons utilisé deux jeux de données dont l'un (Columbia University Image Library (COIL-100)) qui comporte des données plus ou moins homogènes, dans lequel la seule différence entre les images de la même classe, est fait d'une manière manuelle en tournant, rognant, zoomant, ou en ajoutant des bruits. Le deuxième jeux de données (Caltech 101) est composé des données très variées, dans lesquelles, les images de la même classe diffèrent les unes des autres au niveau de la taille couleur, forme, etc.

3. Préparation de la base d'apprentissage

Tout d'abord nous avons effectué une dissection de la base d'image en 2 parties, une partie pour l'entraînement et une pour le test. Puis nous avons fixé la taille des données à 50% pour l'entraînement et 50% pour le test. Dans chaque classe on a deux repertoire, train et test respectivement pour l'entrainement et le test.

4. L'entraînement

Pour l'entraînement de notre modèle nous avons creer une base de référence de descripteur dont dont les étapes sont:

1. Détecter les points d'intérêts avec SIFT
2. Récupérer les descripteurs associés à ce point.
3. Stocker les descripteurs dans une liste [classe][image].
4. Enregistrer dans un seul fichier (toutes les images dans un fichier).

On fait la même chose pour chacune des images du dossier **train** de chaque classe.

4.1 Détection des points d'intérêts

Pour utiliser SIFT dans opencv, on crée l'objet SIFT et qui prend en paramètre le nombre de point maximale à détecter. Il existe parfois des images qui ont plus de point d'intérêt et d'autre ont moins, ce qui influence sur le temps de traitement de toute la base. Pour la normalisation nous avons fixé le nombre de point détecté pour chaque image.

4.2. Récupérer les descripteurs associés à ce point

Pour obtenir les descripteur avec les points dans opencv nous avons fait detectAndCompute. Puis on les stocke dans un tableau à double dimension [classe][image]. Exemple : pour 15 classes avec 36 images d'entraînement on aura 15x36= 540.

4.3. Sauvegarde des données

Pour l'enregistrement de nos données d'entraînement, nous avons utilisé la sérialisation pour conserver les états de notre tableau.

5. Le test

Pour pouvoir tester notre programme, on parcourt le dossier test de chaque classe, et pour chaque image on fait :

1. Récupère ses points d'intérêt
2. Pour fait le Matching avec chacune image dans la base.
3. On trie les images de la base avec le nombre de correspondance de point d'intérêt.
4. En fonction du nombre de voisin le plus proche qu'on veut avoir on prends les n premiers éléments du tableau trié.
5. On prend la classe dominante comme la classe de l'objet en entrée
6. Si ça correspond à la classe de cette dernière c'est vrai sinon c'est faux

5.1. Le matching avec chacune des images dans la base

Le matching consiste à comparer les points d'intérêt entre deux images données. Pour affirmer qu'un point d'intérêt X d'une image A et correspond à un point dans une autre image B, on calcule le carré de distance entre ce point (de A) avec tous les point dans B et on trie puis prends 2 premiers points Y et Z qui ont le carré de distance plus petit. Si la différence entre XY et XZ est inférieur à 0.6 on dit que celui qui a la plus petite distance avec X est le point correspondant de X dans dans B, sinon il n'y a pas de correspondance. Pour faire ça dans opencv il existe plusieurs algorithmes de KNN implémenté comme le Flann KNN, BrutForce, BestMatcher, et on n'a pas testé plusieurs algorithmes de ce type mais on a directement le flannKNN based Matcher.

5.2. Création du tableau trié sur le nombre de correspondance

Pour chaque image qu'on trouve dans les répertoires tests nous essayons de faire la correspondance et on compte le nombre de correspondance avec chaque image, et nous trions à base de celle-ci pour avoir un tableau de plus grand vers un plus petit. Nous pouvons alors prendre après le nombre de voisin le plus proche que nous voulons mais dans notre cas nous avons effectué plusieurs expérimentations.

5.3. Choix de la classe d'appartenance

Avec les n voisins les plus proches nous avons une liste contenante peut être de différents types, et nous prenons la classe la plus répétées dans cette liste pour prédire la classe de l'image en entrée.

5.4. La matrice de confusion

la matrice de confusion est une matrice qui mesure la qualité d'un système de classification. Chaque ligne correspond à une classe réelle, chaque colonne correspond à une classe estimée, cela nous permet de voir et mesure la précision de notre prédiction. Pour avoir ce tableau, on a créé un tableau F de taille [nbclasse][nbclasse] double dimension avec la taille de nombre de classe de chaque. On initialise ses valeurs à 0 au début du test. Après chaque image on prend la classe de l'image qui est c et la classe prédite c1. On incrémente la valeur à l'emplacement F[c][c1] +=1. Par
exemple si nous avons 3 classes, le tableau F va être :
[0,0,0],
F = [0,0,0]
[0,0,0]
et après avoir passé une image de classe 1, on a prédit la classe 2, le tableau devient alors.
[0,0,0],
F = [0,0,1],
[0,0,0]
et ainsi de suite.
Pour la visualisation, nous avons transformé ce tableau en une matrice et avons dessiné chaque valeur par une couleur en fonction du nombre de correspondance.

5.5. Evaluation du modèle

Pour pouvoir évaluer notre modèle de prédiction nous avons utilisé le pourcentage des images bien classifiée sur le nombre totale des images c'est a dire:
Taux de correction = Nombre de prédiction correcte / Nombre totale des images

6. L'expérimentations

Nous avons varier différent paramètre de notre modèle, pour l'expérimentation:

- Nous avions utiliser les 2 bases d'image afin de pouvoir faire un comparaison a la fin des résultats. les base sont nommées dataset et dataset1;
- Nous avons nommé notre descripteur **d** que nous avons testé avec les différentes valeurs {16,32,64}, pour voir comment ça vas affecter le résultat de prédiction.
- Nous avons nommée le nombre de voisin le plus proche k lors de la prédiction puis nous l'avons attribuer les valeur { 3,6,12}, comme on a pas la valeur de paramètre la plus adapté et laquelle montre plus de bon résultat.
- Nombre de classe N = {15 classes}

Et on affiche la matrice de confusion pour le paramètre qui donne le meilleurs résultat pour chaque base. On recueille les résultats issus de chaque expérimentation et met dans un tableau on crée un graphe pour voir une interprétation à celui-ci.

6.1.Expérimentation 1 :

Base b = 1 ; dataset1
N= 15 classes

Descripteurs d = 16

K Voisins	Taux de correction%
3	13.83 %
6	15.06 %
12	14.81 %

Descripteurs d = 32

K Voisins	Taux de correction%
3	17.28 %
6	15.8 %
12	17.28 %

Descripteurs d = 64

K Voisins	Taux de correction%
3	18.77 %
6	17.28 %
12	16.79 %

6.2.Expérimentation 2 :

Base b = 2 ; dataset
N= 15 classes

Descripteurs d = 16

K Voisins	Taux de correction%
3	61.78 %
6	62.34 %
12	60.85 %

Descripteurs d = 32

K Voisins	Taux de correction%
3	72.36 %
6	72.54 %
12	73.65 %

Descripteurs d = 64

K Voisins	Taux de correction%
3	76.25 %
6	75.14 %
12	74.95 %

7. RÉSULTATS

Les deux meilleurs résultats des deux expérimentations sont :

7.1.Expérimentation 1

k = 3 avec descripteur d = 64 avec le taux : 18.77 %

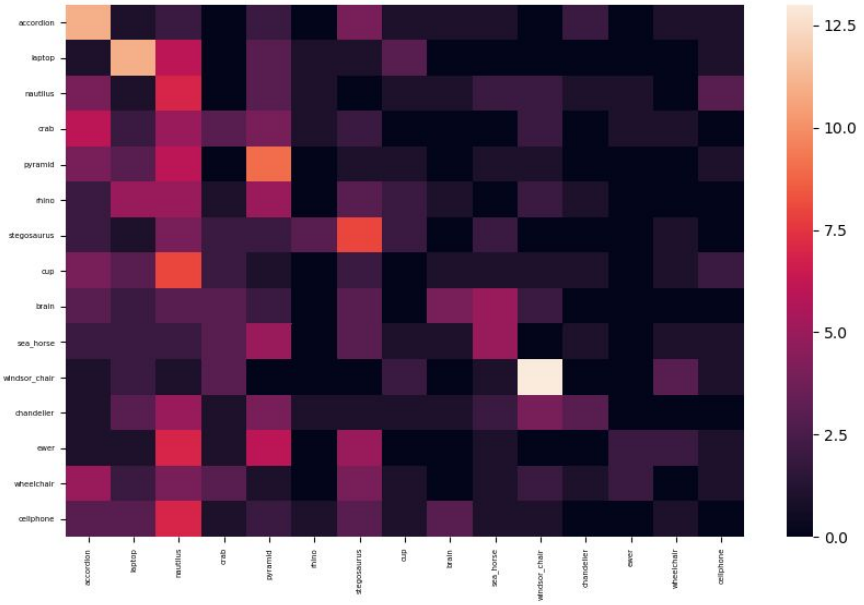

Figure 2 : Matrice de confusion k = 3 , d = 64

7.2. Expérimentation 2

k = 3 avec descripteur d = 64 avec le taux : 76.25 %

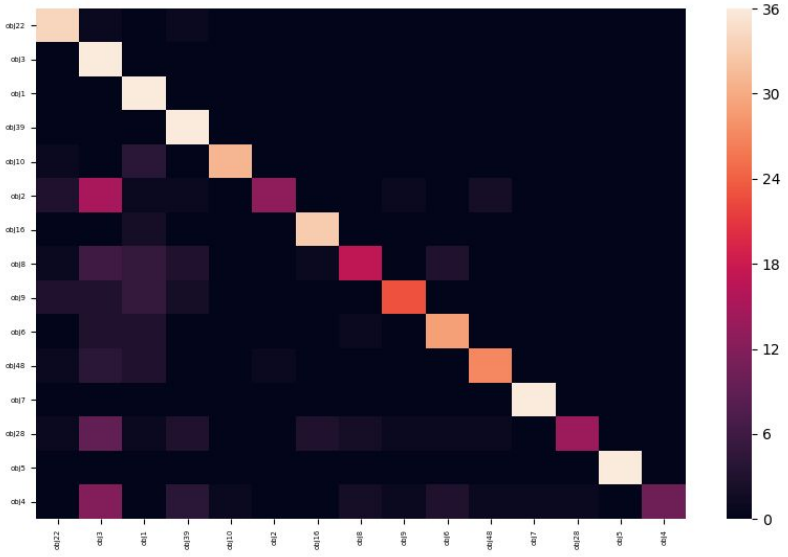

Figure 3 : Matrice de confusion k = 3 , d = 64

8. INTERPRÉTATION ET ANALYSE DES RÉSULTATS

Pour la première base c'est à dire **dataset1**, nous avons comme résultats 18,77% de taux correction au maximum, cela est due aux différences naturelles des images, c'est-à-dire les différences entre les images ne sont pas dues aux transformations manuelles telles que les rognages, rotations, zoom, etc, elles ont donc des tailles, de forme, sources différentes. Et ça résulte que notre modèle distingue mal les images de différentes classes, comme montre la matrice de confusion dans le résultat de l'expérimentation 1. Il y a des images qu'il arrive à bien classifié et d'autres non. Nous avons essayé de voir la matrice de plus près les images qui ont plus de confusion entre eux et on a remarqué que vraiment ces images ont une ressemblance au niveau de la forme par exemple la classe wheelchair et windsor_chair (images ci-dessous) il y a une forte confusion entre eux :

Figure 4 : windor_chair --- wheelchair

Pour la deuxieme base (dataset), nos résultats sont à 76% de taux correction au maximum, cela est normale car les images au sein de la même classe sont déformées par exprès, avec les rognages, rotations, agrandissement, réduction, ajout des bruits (comme expliqué par le propriétaire du dataset. Et d'où l'image dans le dossier d'entraînement et le test sont les mêmes images mais rajouté des modifications très légères. Et on a aussi une matrice de confusion assez bien car on a presque la diagonale en couleur très claire ce qui signifie un meilleur taux de correction.

Remarque :
Si on ne limite pas le nombre de descripteur (si on ne normalise pas) on aura trop de décalage dans les données d'apprentissage stockées. Ça peut aussi causer la lenteur des tâches, car parfois SIFT détecte jusqu'à plusieurs centaines voir de millier sur une même image en fonction de la taille et la qualité de l'image d'après SIFT. Plus de descripteur c'est plus lent mais augmente la précision et moins de

descripteur plus rapide mais affecte plus ou moins le résultat et il y a un très fort décalage comme on a vu dans nos graphiques et matrice de confusion et lorsqu'on parle de descripteur ici c'est le descripteur associé au point d'intérêt détecté.

CONCLUSION

Au cours de ce Tp comme le but nous le demandait nous avons réalisé une application qui consiste à détecter les objet dans une image en utilisant SIFT. Pour ce faire l'application prend en entrée une image et donne en sortie le nom de l'objet contenu sur l'image. En sachant que cela a été faite en utilisant une base d'image, et afin de voir l'efficacité de SIFT on a utilisé deux bases qui sont complètement différentes. Une première qui contient des images totalement variées de même type, et une autre contient des images de même classes très homogène. D'après nos résultats, on peut conclure que SIFT est une méthode très robuste pour l'utilité dans la reconnaissance d'objet. Elle donne un résultat moins bon sur des images de même type d'objet mais acquise d'une manière, formes, tailles, traitements très différents.

Référence

[1] https://www.cs.ubc.ca/~lowe/keypoints/
[2] https://fr.wikipedia.org/wiki/Matrice_de_confusion
[3] Feature Matching opencv
https://docs.opencv.org/3.4/dc/dc3/tutorial_py_matcher.html
[4] Dataset1 (base
2)http://www1.cs.columbia.edu/CAVE/software/softlib/coil-100.php
[5] Dataset (base 1)http://www.vision.caltech.edu/Image_Datasets/Caltech101/
[6]https://openclassrooms.com/fr/courses/4470531-classez-et-segmentez-des-donnees-visuelles/5053196-decrivez-efficacement-les-features-detectees-avec-sift