



Université Nationale du Vietnam

Institut Francophone International

TRAVAUX PRATIQUES DE VISION PAR ORDIANATEUR

Reconnaissance d'objets avec le descripteur SIFT

Rédigé par :

AMALA Gane Kwada et SANON Dô Ambroise Judicaël

Supervisé par :

M. NGUYEN Thi Oanh

Enseignante à IFI

INTRODUCTION

La vision par ordinateur est le domaine de l'informatique qui permet aux ordinateurs de voir, d'identifier et de traiter des images de la même manière que la vision humaine, puis de fournir un résultat approprié. Les tâches de vision par ordinateur comprennent des méthodes pour acquérir, traiter, analyser et comprendre des images numériques et extraire des données de grande dimension du monde réel afin de produire des informations numériques ou symboliques, par exemple sous la forme de décisions.

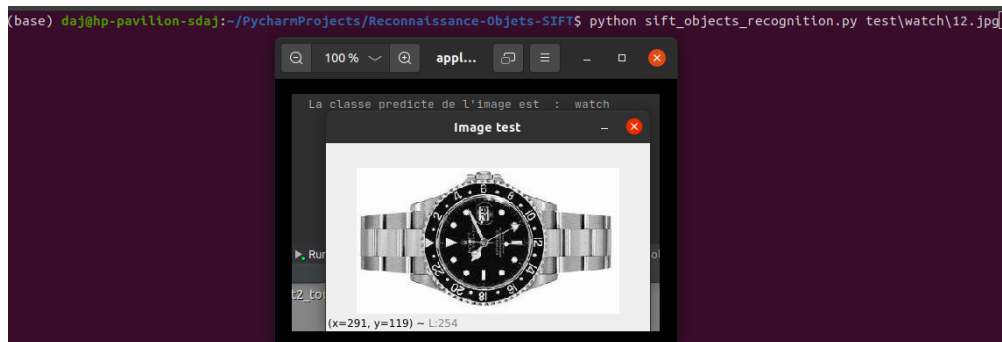
Le présent document fait office de rapport présentant les différentes étapes que nous avons effectuées pour apporter une solution au problème de la reconnaissance d'objets avec le descripteur SIFT. Ce travail s'inscrit dans le cadre du cours de Vision Par Ordinateur, dispensé à l'Institut Francophone International.

1 Présentation de l'application

Environnement de développement :

- Python version 3.7
- OpenCV 4.4.0.46
- Pycharm et Google colab pour certains tests.

Notre programme fonctionne comme suit : A partir de l'invite de commande, exécutez : **sift_objects_recognition.py** avec pour paramètre une image. **sift_detector.py** suivi d'une image pour une demo de l'algorithme de sift sur une seule image. **split.py** permet de diviser la base de données en pourcentage égale (test et train). **matrix_confusion.py** permet de visualiser la matrice de confusion.



2 Acquisition des images

Pour réaliser ce travail, nous avons utilisé la base d'images Caltech 101 dont le lien ci-dessous a été fourni dans la description du TP. Ce jeu de données contient environ 101 catégories d'objets et chaque catégorie contient environ 50 images ayant chacune une taille de 300×200

http://www.vision.caltech.edu/Image_Datasets/Caltech101/

Nous rappelons que pour respecter la description du TP, nous n'avons scindé notre jeu de données en deux parties : 50% pour l'entraînement et 50% pour le test.

3 Extraction des caractéristiques avec l'algorithme SIFT pour une image

L'algorithme des SIFT (scale-invariant feature transform ou transformation de caractéristiques visuelles invariantes à l'échelle en français), est un algorithme développé par David Lowe en 1999. L'idée général de cette méthode est de transformer une image en vecteurs de caractéristiques, lesquels doivent être dans l'idéal invariants aux transformations géométriques (rotation, mise à l'échelle). Il est composé de deux grandes étapes : le calcul des points d'intérêts (figure 2) et des descripteurs et la mise en correspondance (figure 3).



FIGURE 1 – Image Crabe

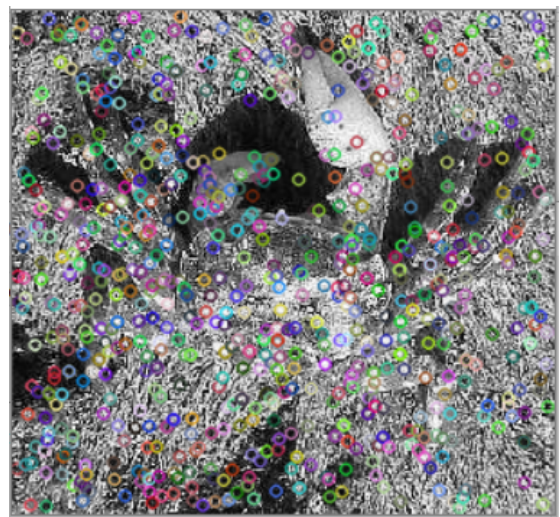


FIGURE 2 – Application de SIFT

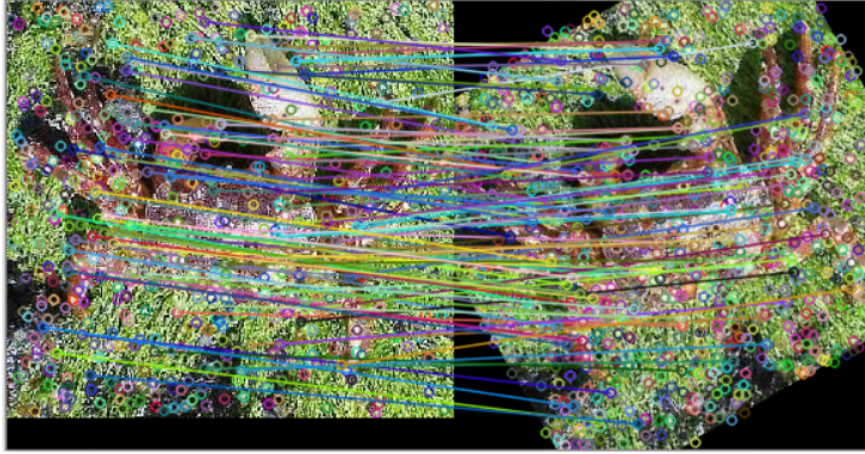


FIGURE 3 – Correspondance entre les éléments caractéristiques après rotation de l'image originale

À travers la figure 3, on remarque que malgré la rotation de l'image, SIFT parvient à établir une correspondance entre l'image originale et celle soumise à une transformation. On constate également qu'au niveau de la figure 2, il y a beaucoup de points d'intérêts mais lorsque nous faisons la correspondance entre eux, tous ces points d'intérêts ne sont pas pris en compte.

4 Préparation de données

Comme nous l'avons indiqué ci haut, notre jeu de données a été divisé en données d'entraînement et de test.

```
[6] split_data(path_image, 0.5)

101 images copiées vers data/train/Leopards
99 images copiées vers data/test/Leopards
33 images copiées vers data/train/lotus
33 images copiées vers data/test/lotus
33 images copiées vers data/train/elephant
31 images copiées vers data/test/elephant
210 images copiées vers data/train/Faces_easy
235 images copiées vers data/test/Faces_easy
19 images copiées vers data/train/gramophone
32 images copiées vers data/test/gramophone
17 images copiées vers data/train/octopus
18 images copiées vers data/test/octopus
24 images copiées vers data/train/stegosaurus
35 images copiées vers data/test/stegosaurus
36 images copiées vers data/train/soccer_ball
28 images copiées vers data/test/soccer_ball
234 images copiées vers data/train/BACKGROUND_Google
233 images copiées vers data/test/BACKGROUND_Google
42 images copiées vers data/train/llama
36 images copiées vers data/test/llama
19 images copiées vers data/train/binocular
14 images copiées vers data/test/binocular
16 images copiées vers data/train/panda
22 images copiées vers data/test/panda
37 images copiées vers data/train/trilobite
49 images copiées vers data/test/trilobite
```

FIGURE 4 – Séparation du jeu de données en deux catégories : Entraînement et Test

5 Phase d'entraînement

Nous avons utilisé l'algorithme SIFT pour détecter les points d'intérêts des images grâce à l'objet `cv2.xfeatures2d.SIFT_create()` de `openCV`. La fonction `detectAndCompute` nous permet de calculer ces points et les descripteurs associés. Nous constatons certaines images présentent plus de points d'intérêts que d'autres. C'est un facteur qui influence le temps de traitement de toute la base de données (figure 5). Par ailleurs, nous avons fait recours à la stérilisation pour conserver l'état des données d'entraînement.

```

❏ Number of Keypoints Detected In The Training Image beaver/18.jpg:556
Number of Keypoints Detected In The Training Image okapi/18.jpg:557
Number of Keypoints Detected In The Training Image minaret/18.jpg:167
Number of Keypoints Detected In The Training Image rooster/18.jpg:506
Number of Keypoints Detected In The Training Image Faces_easy/73.jpg:429
Number of Keypoints Detected In The Training Image pagoda/18.jpg:387
Number of Keypoints Detected In The Training Image saxophone/18.jpg:547
Number of Keypoints Detected In The Training Image trilobite/18.jpg:1336
Number of Keypoints Detected In The Training Image stapler/18.jpg:160
Number of Keypoints Detected In The Training Image helicopter/18.jpg:230
Number of Keypoints Detected In The Training Image hedgehog/18.jpg:1090
Number of Keypoints Detected In The Training Image wrench/18.jpg:74
Number of Keypoints Detected In The Training Image chair/18.jpg:423
Number of Keypoints Detected In The Training Image airplanes/73.jpg:370
Number of Keypoints Detected In The Training Image brontosaurus/18.jpg:118
Number of Keypoints Detected In The Training Image panda/11.jpg:426
Number of Keypoints Detected In The Training Image flamingo/18.jpg:304
Number of Keypoints Detected In The Training Image dollar_bill/18.jpg:429
Number of Keypoints Detected In The Training Image headphone/18.jpg:355
Number of Keypoints Detected In The Training Image crocodile_head/18.jpg:494
Number of Keypoints Detected In The Training Image joshua_tree/18.jpg:640
Number of Keypoints Detected In The Training Image bonsai/18.jpg:386

```

FIGURE 5 – *Points intérêts de notre base d'images d'entraînement*

Les figures ci dessous, nous montrent quelques images de notre base d'entraînement avec les points d'intérêts calculés par l'algorithme SIFT.

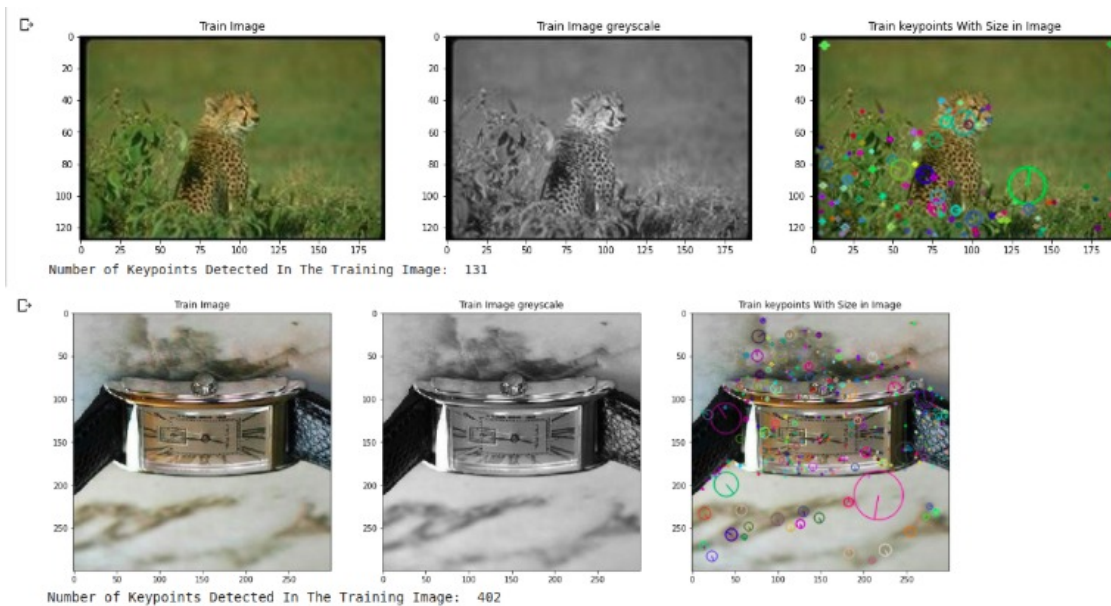
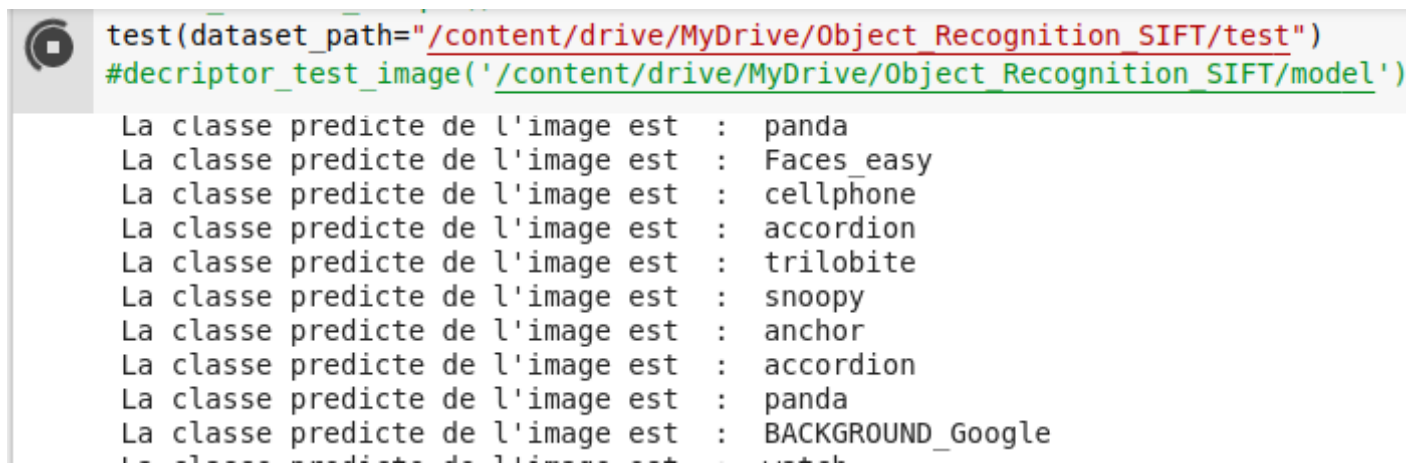


FIGURE 6 – *Entraînement et detection des caracteristiques*

6 Phase de test

Pour chaque image de test, nous calculons les points d'intérêt et les descripteurs associés, ensuite nous faisons la mise en correspondance (matching) avec les images de la base. Pour ce faire nous utilisons l'algorithme KNN(K plus proches voisins) et plus précisément le **FLAN Based Matched** grâce à la fonction `cv2.FlannBasedMatcher()` de openCV. Notre choix s'est porté sur cette methode car elle est plus rapide que les autres comme **BFMatcher** sur les grands ensembles de données et pour les entités de grande dimension. Nous avons fixé notre k à 5. Le choix de la classe d'appartenance de l'image en entrée sera donc celle la plus répétée parmi les 5 plus proches.



```
test(dataset_path="/content/drive/MyDrive/Object Recognition SIFT/test")
#descriptor_test_image('/content/drive/MyDrive/Object Recognition SIFT/model')

La classe predite de l'image est : panda
La classe predite de l'image est : Faces_easy
La classe predite de l'image est : cellphone
La classe predite de l'image est : accordion
La classe predite de l'image est : trilobite
La classe predite de l'image est : snoopy
La classe predite de l'image est : anchor
La classe predite de l'image est : accordion
La classe predite de l'image est : panda
La classe predite de l'image est : BACKGROUND_Google
```

FIGURE 7 – Phase de test

7 Evaluation et Expérimentation

Pour évaluer notre modèle, nous calculons le pourcentage des prédictions correctes sur le nombre total des images.

Nous avons testé plusieurs experiences avec des quantités de données differentes. Notre objectif était entre autre la recherche de bons résultats. Pour ce faire, nous avons modifié quelques paramètres comme le nombre de descripteurs à calculer pour chaque image, la valeur des voisins les plus proches pour le matching, **param_search**(le nombre de fois où les arbres de l'index doivent être parcourus de manière récursive). Des valeurs plus élevées de certains paramètres donnent de meilleure précision, mais prennent également plus de temps, et le taux de rapport et definit par David Lowe dans son article[1].

7.1 Expérimentation 1

Nous utilisons toutes les classes et toutes les images de notre base de données. De plus, nous prenons en compte toutes les caractéristiques calculées par l'algorithme SIFT. On obtient une précision de 2,02%.

Nombre de classe	Nombre de descripteurs	K-voisins	Test de rapport	Search_param	Précision
101	tous	5	0,6	100	2,02%

FIGURE 8 – *Expérience 1*

Notre jeu de données est composé de 101 classes, la matrice de confusion est donc de taille 101*101, avec l'axe de gauche montrant la vraie classe (comme connu dans l'ensemble de test) et l'axe du haut montrant la classe affectée à un élément avec cette vraie classe. Chaque élément i, j de la matrice est le nombre d'éléments avec la vraie classe i qui ont été classés comme étant dans la classe j . Les valeurs sur la diagonale représentent les prédictions correctes.

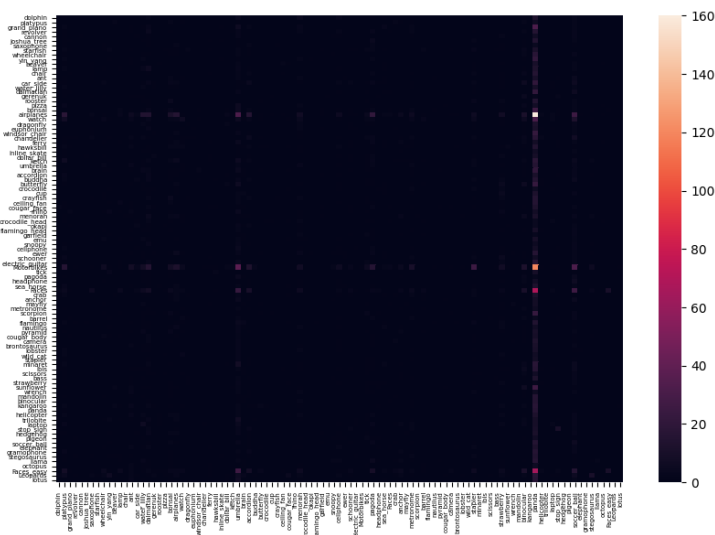


FIGURE 9 – *matrice de confusion experimentation 1*

On remarque une confusion entre les classes **panda** et **airplane**

7.2 Expérimentation 2

Ici, nous fixons le nombre de descripteurs à 100. On obtient une précision faible mais plus élevée que la première : 2,42%.

Nombre de classe	Nombre de descripteurs	K-voisins	Test de rapport	Search_param	Précision
101	100	5	0,6	100	2,42%

FIGURE 10 – *Expérience 2*

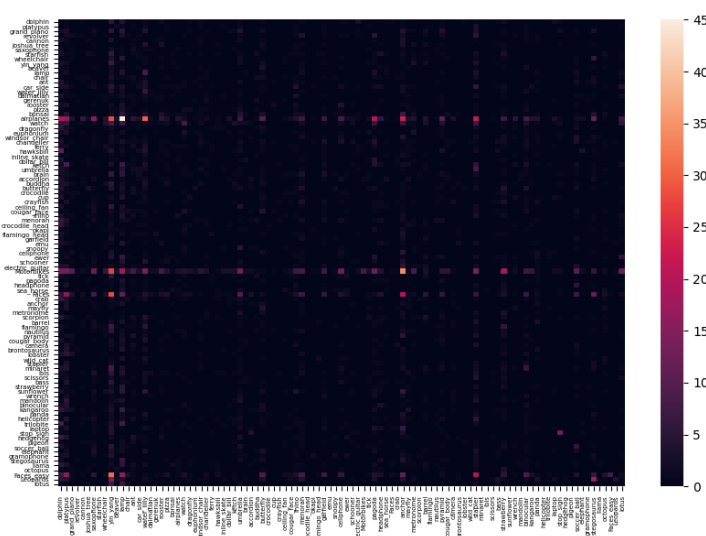


FIGURE 11 – *matrice de confusion expérimentation 2*

La matrice de confusion nous montre que beaucoup d'images de la classe **airplane** ont été classés comme **lamp**.

7.3 Expérimentation 3

Compte tenu des faibles précisions obtenues plus haut, nous réduisons le nombre de classes à 20 (choisies aléatoirement). Cela a permis de réduire le temps de traitement. Nous obtenons une précision de 13,67%.

Nombre de classe	Nombre de descripteurs	K-voisins	Test de rapport	Search_param	Précision
20	150	5	0,7	100	13,67%

FIGURE 12 – *Expérience 3*

A travers cette matrice, nous constatons une dominance de la classe **airplane** qui totalise beaucoup de prédiction correcte.

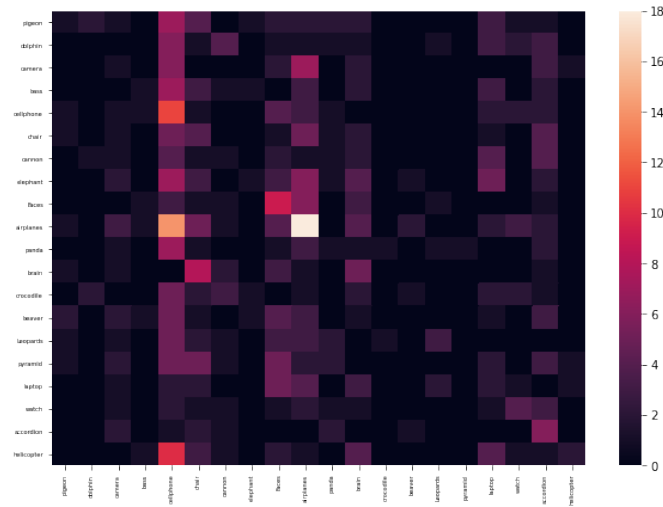


FIGURE 13 – *matrice de confusion expérimentation 3*

7.4 Expérimentation 4

Ici nous sélectionnons deux classes : **airplane** et **watch**. Nous obtenons une précision nettement supérieure : 73,92%.

Nombre de classe	Nombre de descripteurs	K-voisins	Test de rapport	Search_param	Précision
2	200	5	0,7	200	73,92%

FIGURE 14 – *Expérience 4*

La matrice de confusion nous indique que le modèle se trompe rarement.

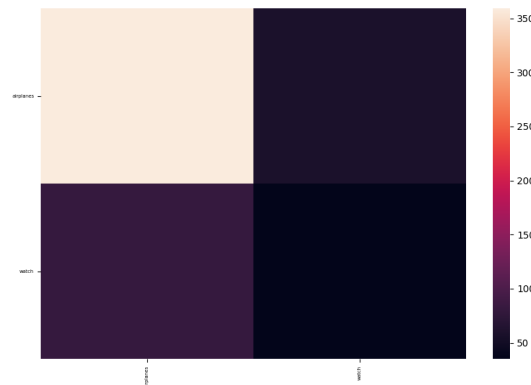


FIGURE 15 – *matrice de confusion expérimentation 4*

7.5 Expérimentation 5

Enfin, nous choisisons deux classes assez similaires et proportionnelles : **faces** **faces_easy**. Nous obtenons une précision de 52,42%.

Nombre de classe	Nombre de descripteurs	K-voisins	Test de rapport	Search_param	Précision
2	200	5	0,7	200	52,42%

FIGURE 16 – *Expérience 5*

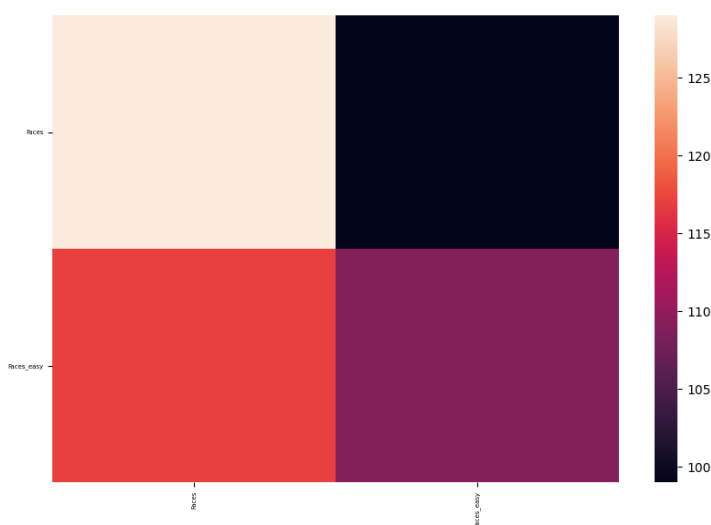


FIGURE 17 – *matrice de confusion experimentation 5*

Effectivement à travers la matrice de confusion, on voit que Le confond assez souvent les deux classes.

8 Interprétation des résultats

Malgré la quantité considérable de données, notre modèle a obtenu une faible précision, il se trompe régulièrement, en affectant un objet a une classe qui n'est pas la sienne. Cela est du a plusieurs facteurs comme :

- la diversité des images, nous avons une base de données avec 101 classes, avec des ressemblances entre elle : **crocidiles** et **crocodiles_head**, **face** et **face_eyes** ;

- Les classes n'ont pas les memes proportions de données par exemple **airplanes=800 images** et **binocular=33 images**. Cela peut biaiser l'interpretation des résultats.
- La classe Google_background qui contient des images diversifiées qui ont des traits de ressemblances avec les autres classes ;
- La qualité des images souvent floues, zoomées, soumises à des transformations(rotation) ;
- Les images ne sont pas prises dans les memes conditions d'eclairage, aussi le contraste est différent pour les images.

Cependant la réduction de classes et l'ajustement de certains paramètres permettent d'obtenir de meilleurs résultats.

9 Quelques prédictions

* Prédictions incorrectes



FIGURE 18 – Quelques Prédictions incorrectes

* Prédictions correctes

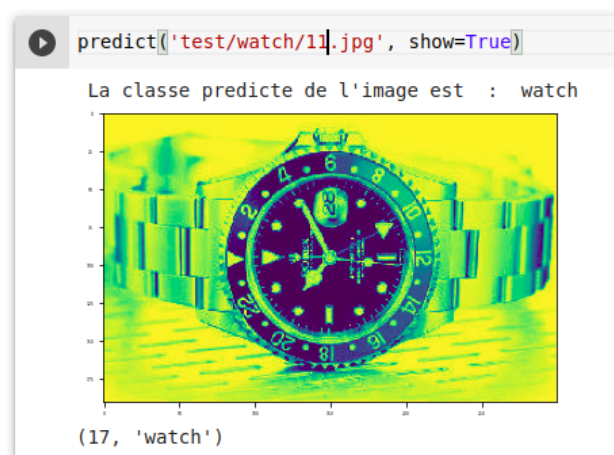


FIGURE 19 – Prédiction correcte

Conclusion

Faire ce mini-projet a été pour nous une opportunité pour mieux assimiler les concepts liés à détection des features notamment l'algorithme SIFT et la notion d'apprentissage, aussi d'appréhender certains concepts théoriques de la Vision par Ordinateur.

Références

- [1] *Distinctive Image Features from Scale-Invariant Keypoints*. David G. Lowe, January 5, 2004. <https://docs.google.com/viewer?a=v&pid=sites&srcid=ZGVmYXVsdGRvbWFpbmxwb3VibGFuZ3NpZnR8Z3g6NjE3MjYxOTcxMWVkOWM2>
- [2] *Introduction à SIFT, Tutotiel OPENCV* https://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_feature2d/py_sift_intro/py_sift_intro.html
- [3] *Caltech 101* http://www.vision.caltech.edu/Image_Datasets/Caltech101/
- [4] *Recognition-of-Objects-using-Shift-detector*, CISSOKO Ben Mamadou <https://github.com/benprano/Recognition-of-Objects-using-Shift-detector>

Lien code source Github : <https://github.com/daj10/SIFT-Objects-Recognition>