RAPPORT
Chef d'Oeuvre
Recette

Courdy-Bahsoun Clémence - Deker Sylvain
Moussa Nahor

Groupe : Ré-id

M2 IGAI

1 Introduction

Le projet proposé par le client, M.Alain Crouzil a pour but le développement d'une interface permettant de faire de la ré-identification de véhicule dans un corpus de vidéo. Il s'agit de détecter les véhicules dans les vidéos et de les comparer au véhicule recherché afin de fournir une liste pertinente des vidéos pouvant contenir le véhicule recherché.

La méthode mise en øeuvre pour la ré-identification de véhicule est appelé Fusion of Attributes and colors features (FACT). Elle repose sur le calcul de trois descripteurs de l'image (texture, couleur et sémantique) utilisé pour calculer un score de distance par fusion unique.
La méthode BOW-SIFT permet de calculer un descripteur de texture pour une image à partir d'un modèle d'apprentissage non supervisé de 10000 classes calculées à partir des descripteurs sift des points d'intérêts des images de la base d'apprentissage. On calcul alors un histogramme à 10000 classes qui compte le nombre de point d'intérêts prédit par classe pour une image, cet histogramme est le descripteur de texture BOW-SIFT
La méthode BOW-CN permet de calculer un descripteur de couleur sous la forme d'un vecteur obtenu par le calcul d'un histogramme sur 250 classes obtenues par un algorithme de k-moyenne. Les descripteurs permettant de calculer l'histogramme sont les valeurs des canaux des pixels.
Le réseau de neurones convolutif GoogleNet prend une image en entrée et en lui appliquant des convolutions il permet d'obtenir différents descripteurs qui permettront une caractérisation de l'image propre au réseau : les poids de la couche finale du réseau vont être utilisés comme descripteur sémantique du véhicule.

Ce rapport contient un manuel utilisateur détaillant les installations préalables et le fonctionnement de l'interface. Puis il présente les différentes fonctions, ainsi que les jeux de tests menés pour valider les résultats. Enfin une rétrospective est faite mettant en avant les points forts du logiciel ainsi que les améliorations possibles.

2 Manuel Utilisateur

Installations préalables

- pip install opencv-python==3.4.2.16
- pip install opencv-contrib-python==3.4.2.16
- pip install pandas
- pip install scipy
- pip install sklearn
- pip install pickle
- pip install keras
- Charger ou cloner le projet `https://github.com/SylvainDeker/Vehicle-ReIdentificatio`
- charger dans le repertoire ../data/Veri_with_plate
  depuis `https://github.com/VehicleReId/VeRidataset`
- Dans *VehicleReId/data/ressources* décompresser le fichier *listeDesBOWSIFT.zip*
- Se placer dans *VehicleReId/GUI* et lancer en console **python2.7 vraiUi.py**

Remarque : Le protocole présenté permet de lancer le logiciel de ré-identification à partir des modèles que nous avons pré-calculés. Cependant les fonctions de calculs des différents descripteurs et les fonctions d'entraînements sont disponibles, et des exemples d'utilisations sont décrit dans les fichiers de tests.

Démarrage

Au lancement du programme l'interface est vierge.

FIGURE 1 – Interface initiale

L'utilisateur ne peut que cliquer sur le bouton "Sélectionner un véhicule".

Ce faisant une fenêtre s'ouvre pour permettre de choisir un véhicule, seul les images au format *.jpg sont prisent en compte.

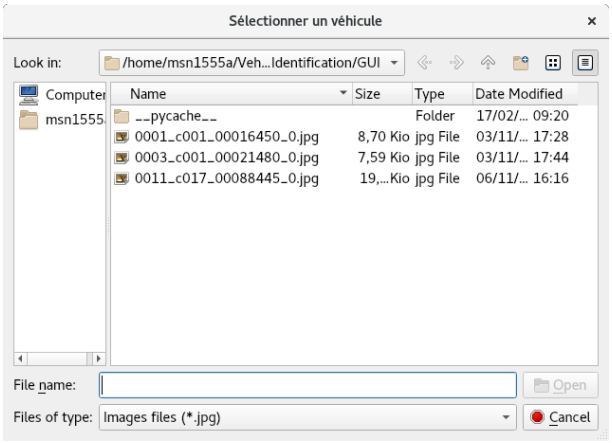

FIGURE 2 – Interface de sélection d'un véhicule

Lancement de ré-identification et résultat

Une fois la sélection effectuée la fenêtre principale réapparaît avec un aperçu du véhicule sélectionné. Il devient également possible de lancer la ré-identification ou de changer d'image.

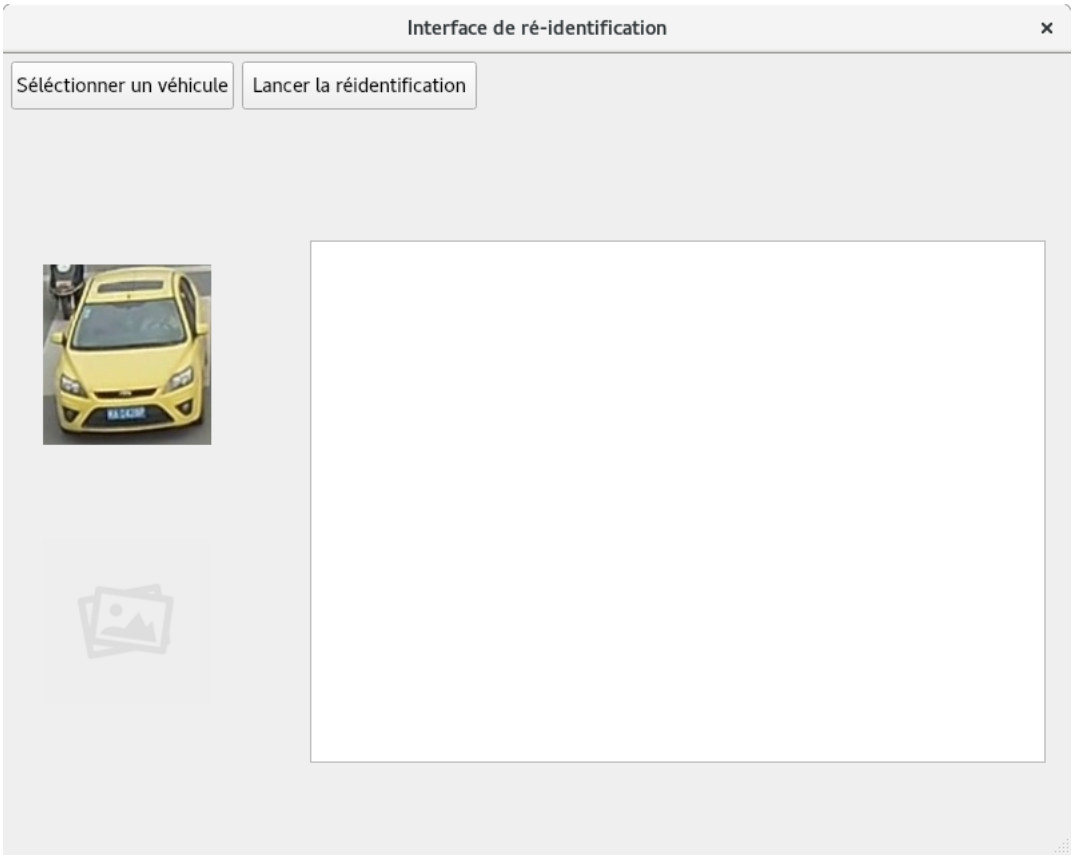

FIGURE 3 – Interface de lancement de la ré-identification

Une fois la ré-identification lancée et terminée les meilleurs résultats s'afficheront dans le tableau. Il est possible de voir les véhicules ré-identifiés en leurs cliquant dessus.

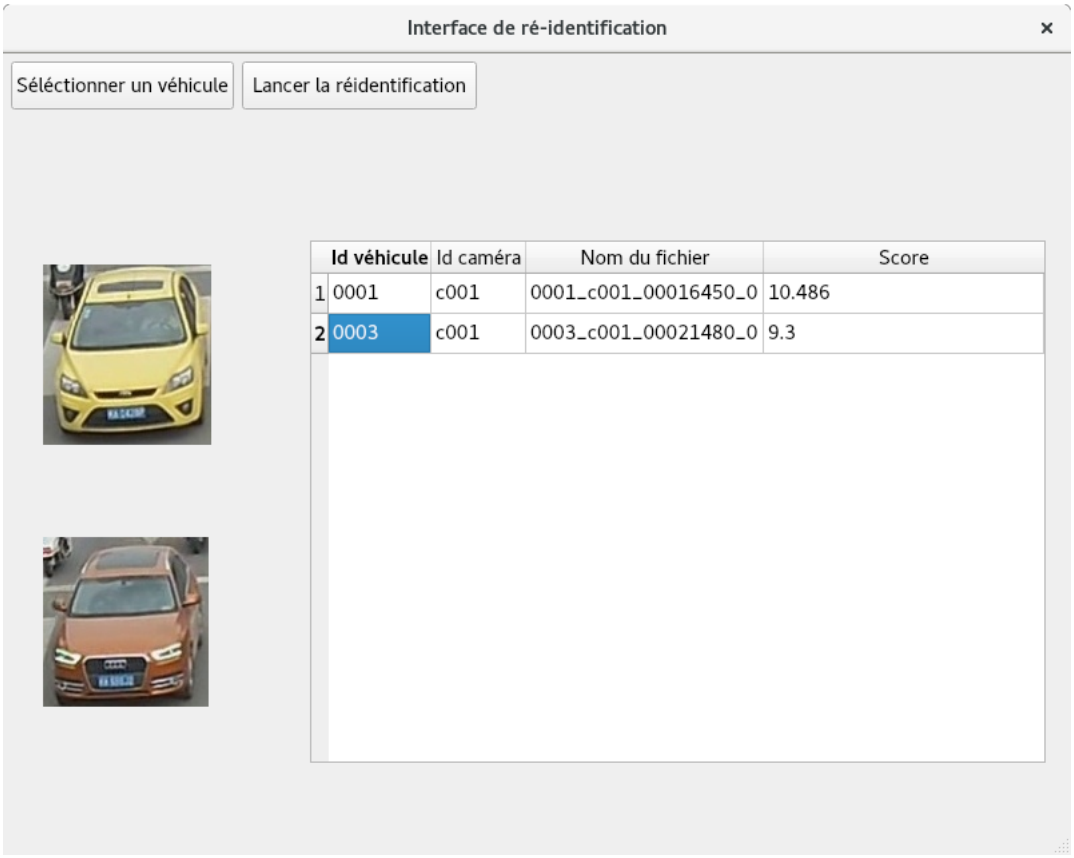

FIGURE 4 – Interface de liste de véhicules ré-identifiés factices pour la démonstration.

3 Descriptifs du code

Interface

Librairie :

Une seule librairie a été utilisé pour réaliser l'interface : PyQt5. Il s'agit d'un module libre qui permet de lier le langage Python avec la bibliothèque Qt.

Les fonctions :

1. **changeLabelImage(widget,fileLocation) → None** :
   Cette fonction prend en paramètres : un label et le chemin absolue d'une image. Elle modifie le l'arrière plan du label avec l'image.
2. **initTableResult(table,n) → None** :
   Cette fonction prend en paramètres : un QTableWidget et un entier. Elle initialise le tableau avec un nombre de ligne correspondant à l'entier, c'est à dire le nombre de résultats que nous voulons afficher. Elle initialise également les colonnes et désactive l'édition des valeurs dans le tableau.
3. **fillLineTable(table,line,data0,data1,data2,data3) → None** :
   Cette fonction prend en paramètres : un QTableWidget , un entier et 4 données. Elle initialise la ligne passée en paramètre avec les données.
4. **openFileNameDialog() → None** :
   Cette fonction est liée au bouton de sélection. Elle affiche la fenêtre de sélection et gère l'affichage de la miniature ainsi que l'activation du bouton de ré-identification.
5. **reIdentificationPlaceHolder() → None** :
   Cette fonction est liée au bouton de ré-identification. Elle lance la ré-identification puis initialise le tableau à partir des résultats et le remplit.
6. **displayResultImage() → None** :
   Cette fonction est celle qui gère l'affichage du résultat en cliquant dans le tableau.
7. **MainWindow() → Interface** :
   Cette fonction est le constructeur de la fenêtre. Il initialise l'affichage et les connexions entre les boutons et les fonctions.

Descripteurs Fact

Librairies :

1. **opencv** : pour lire les images, et pour calculer les descripteurs sift des points d'intérêts d'une image. Le module *sift* de opencv n'est plus libre de droit dans les dernières versions d'opencv, il faut donc travailler avec la version 3.4.2.16 et sous python 2.17.
2. **pickle** : pour charger le modèle de k-moyenne
3. **numpy** : pour travailler sur des tableaux à 2 dimensions.

Les fonctions :

1. **calculerFact(scoreCN,scoreSift,scoreGoogleNet) → float** :
   Fonction qui à partir des scores de distances euclidiennes entre les descripteurs de sémantique, de couleur et de texture de l'image recherchée et celle de la base d'apprentissage, calcule le score final comme la somme pondérée des 3 scores. Les scores sont pondérés à 0,7 pour le score sémantique, 0,2 pour le score de couleur et à 0,1 pour le score de texture.
2. **listerScoresFact(imageCherchee,listeImagesRef,listeDesBOWSIFT,featureExtract → list<(string,float)>** :
   Fonction qui va calculer le score FACT par l'appel à la fonction *calculerFact(scoreCN,scoreSift,scoreGoogleNet*, $\forall i \in [0, len(listeImageRef)$, on créée alors un tuple (listeImageRef[i],score) que l'on ajoute à la liste qui sera retournée en sortie de la fonction lorsque tous les scores fact auront été calculé.
3. **trierListeCroissante(listeATrier) → list<(string,float)>** :
   Fonction qui à l'aide la fonction de trie déjà existante *sorted* va permettre de trier la liste de tuple en fonction du deuxième arguments de chaque tuple le score et ainsi d'obtenir la liste des tuples (nomImageRef,scores) triée par ordre croissant de distance.

Descripteurs Sémantiques

Librairies :

1. **keras** : pour le calcul du modèle d'apprentissage supervisé par réseau de neurones convolutif googleNet

Les fonctions :

1. **prepareSemanticDataSet(featureExtractor) → None** :
   Cette fonction est une fonction de prétraitement : elle calcule et enregistre les descripteurs sémantiques de chaque image d'entraînement . Ces descripteurs servent ensuite à faire la comparaison avec le véhicule à re-identifier. Elle prend en paramètre un modèle Keras qui renvoie les valeurs de l'avant dernière couche du réseau GoogleNet.
2. **googleNetScore(imgPath,featureExtractor) → list<(float,string)>** :
   Fonction qui retourne une liste contenant, pour chaque image d'entraînement, le score de distance entre le descripteur sémantique de l'image *imgPath* prédit grâce au modèle *featureExtractor*, et tous les descripteurs sémantiques des images de la base d'apprentissage pré-calculés.

Descripteurs Textures

Librairies :

1. **opencv** : pour lire les images, et pour calculer les descripteurs sift des points d'intérêt d'une image. Le module *sift* de opencv n'est plus libre de droit dans les dernières versions d'opencv, il faut donc travailler avec la version 3.4.2.16 et sous python 2.17.
2. **pandas** : pour construire une trame de données à partir du dictionnaire des descripteurs sift.
3. **sklearn** : pour calculer le modèle d'apprentissage par k-moyenne, et prédire les labels des pixels d'intérêt.
4. **pickle** : pour mémoriser le modèle dans un fichier binaire.
5. **scipy** : pour calculer la distance euclidienne entre deux descripteurs BOW-SIFT.
6. **numpy** : pour travailler sur des tableaux à 2 dimensions plutôt que sur des listes.
7. **os** : pour compter le nombre de fichier d'un répertoire.

Les fonctions :

1. **descripteurSift(cheminImage) → np.array<float>** :
   Fonction qui retourne les descripteurs sift des pixels d'intérêts d'une image
2. **creerFichiersDico(nomsImage,cheminImage,cheminRepDes,indDIm, indFIm) → void** :
   Fonction qui crée pour les *indFIm* premières images de *cheminImage/nomsImage* les *indFIm* fichiers contenant les descripteurs sift. Un fichier de de *cheminRepDes* contient les descripteurs de l'image auquel il est associé.
   Cette fonction a été mise en place car la base d'apprentissage étant de 37746 images, il faudrait un GPU pour calculer l'intégralité des descripteurs BOW-SIFT en un temps raisonnable. De plus l'utilisation du module sift n'étant pas possible avec toutes les versions d'opencv, on peut alors mettre en øeuvre le calcul du modèle d'apprentissage à partir de ces fichiers depuis google collabotary en simulant un GPU.
3. **extraireDico(cheminRepDes,nomsFichierDes,indDIm,indFIm,pas) → np.array<float>** :
   Cette fonction extrait les *indFIm/pas* descripteurs sift de la liste des fichiers *nomsFichierDes* contenu dans le repertoire *cheminRepDes* et les concatène

en un tableau qui constitue le dictionnaire des descripteurs qui sera utilisé pour le calcul du modèle d'apprentissage (Bag-Of-Word par k-moyenne)

4. **calculerClassesDescripteurs(dicoDescripteur,k) → string** :
   Fonction qui va calculer le modèle d'apprentissage par k-moyenne sur les données *dicoDescripteur* avec k classes, et enregistrer ce modèle sous forme binaire dans le fichier *data/ressources/modeleBOWSIFT.pkl*. Le chemin de ce fichier est retourné par la fonction
5. **calculerHistogrammeImage(cheminImage,kmeans,k) → np.array<int>** :
   Fonction qui calcule le descripteur BOW-SIFT de l'image *cheminImage* à partir du modèle *kmeans* préalablement chargé. C'est à dire que la sortie est un tableau à 1 ligne et k colonnes (correspondant aux classes du modèle). La valeur contenue dans chacune des k occurrencess correspondent aux nombres de pixels d'intérêts de l'image ayant été prédit comme appartenant à chaque classe.
6. **calculerHistogrammeEntrainement(nomsImage,cheminImage, cheminRepDesBOWSIFT,kmeans,k,nbImg) → void** :
   Cette fonction permet de calculer les *nbImg* descripteurs BOW-SIFT des images de *cheminImage/nomsImage[i]* avec $i \in [0, nbImg]$ et de les enregistrer dans des fichiers (1 fichier par descripteur BOW-SIFT) contenu dans le répertoire *cheminRepDesBOWSIFT*.
7. **extraireDesBOWSIFT(cheminRepDesBOWSIFT, nomsFichierBOWSIFT,indDIm,indFIm) → list<np.array<float»** :
   Cette fonction fait une liste des descripteurs BOW-SIFT de la base d'apprentissage. On traite les *cheminRepDesBOWSIFT/nomsFichierBOWSIFT[i]* avec $i \in [indDIm, indFIm]$.

Descripteurs Couleurs

1. **Extraction des descripteurs** Dans un premier temps, les bordures des images sont supprimées (la largeur de la bordure à supprimer est proportionnelle à la taille de l'image). Ceci permet de s'assurer que la majorité des pixels présent sur la photo soient ceux de la voiture, chose importante pour la détection de la couleur du véhicule car la couleur est déterminée à partir de la couleur majoritaire sur la photo. Si ce pré-traitement n'a pas lieu la couleur grise de la route se trouverait (dans une bonne partie des cas) majoritaire.
   Ensuite les images se voient appliquer un filtre moyenneur dont la taille du noyau est du même ordre de grandeur que le sous-échantillonage qui suit. L'intérêt de rendre flou l'image est d'obtenir une disparité moindre des groupes de couleurs, ce qui rend l'opération de K-moyenne plus efficaces.
   Finalement, le sous-échantillonage de la photo permet de réduire drastiquement le nombre de donnée à traiter. L'application du flou précédemment expliqué permet aussi de garder l'influence des pixels perdus sur les pixels échantillonnés. Ce qui permet de réduire de façon importante le nombre de pixels sans trop perdre la qualité d'information.
   L'image résultante est le descripteur de l'image, c'est un ensemble de pixel qui caractérise une image. Sur la figure 5, la colonne de gauche représente les images d'origine et la colonne de droite représente les images une fois traitées.
2. **Filtrage des descripteurs pertinents** Les descripteurs obtenus précédemment sont ensuite classés par groupe à l'aide d'un classifieur par k-moyenne où k=8. A ce niveau le modèle sert uniquement à séparer les différentes classes de couleur de l'image, c'est pourquoi les données (c'est à dire le descripteur) d'entraînement du k-moyen sont les mêmes que pour la prédiction (on a une instance de kmean par image). Sur la colonne de droite de la figure 6 (et la colonne de gauche de la figure 7) est affiché une représentation 3D (selon rouge,vert,bleu) des descripteurs qui leur est associé sur la colonne de gauche.

3. **Détection de la couleur du véhicule** Une fois la classification et l'étiquetage des données réalisé, l'histogramme des classes permet de déterminer celle représentant la couleur avec le plus d'individu. C'est cette classe qui est représentative de la couleur de la voiture. La couleur résultante est la moyenne des pixels dont le label correspond à la classe dominante. Sur la colonne du milieu de la figure 7 se trouve les pixels de la classe dominante. La troisième colonne représente la couleur de la voiture obtenue. C'est cette couleur qui sert de référence pour entraîner le modèle qui servira à la prédiction de correspondance des véhicules.

FIGURE 5 – Extraction des descripteurs

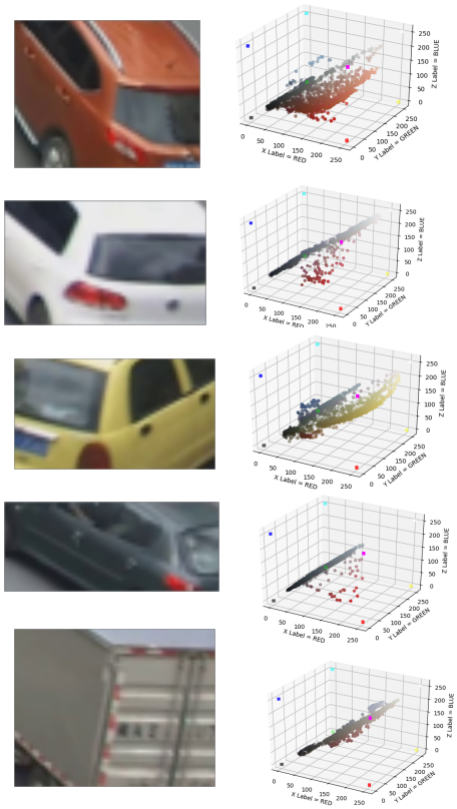

FIGURE 6 – Detection de la couleur du véhicule

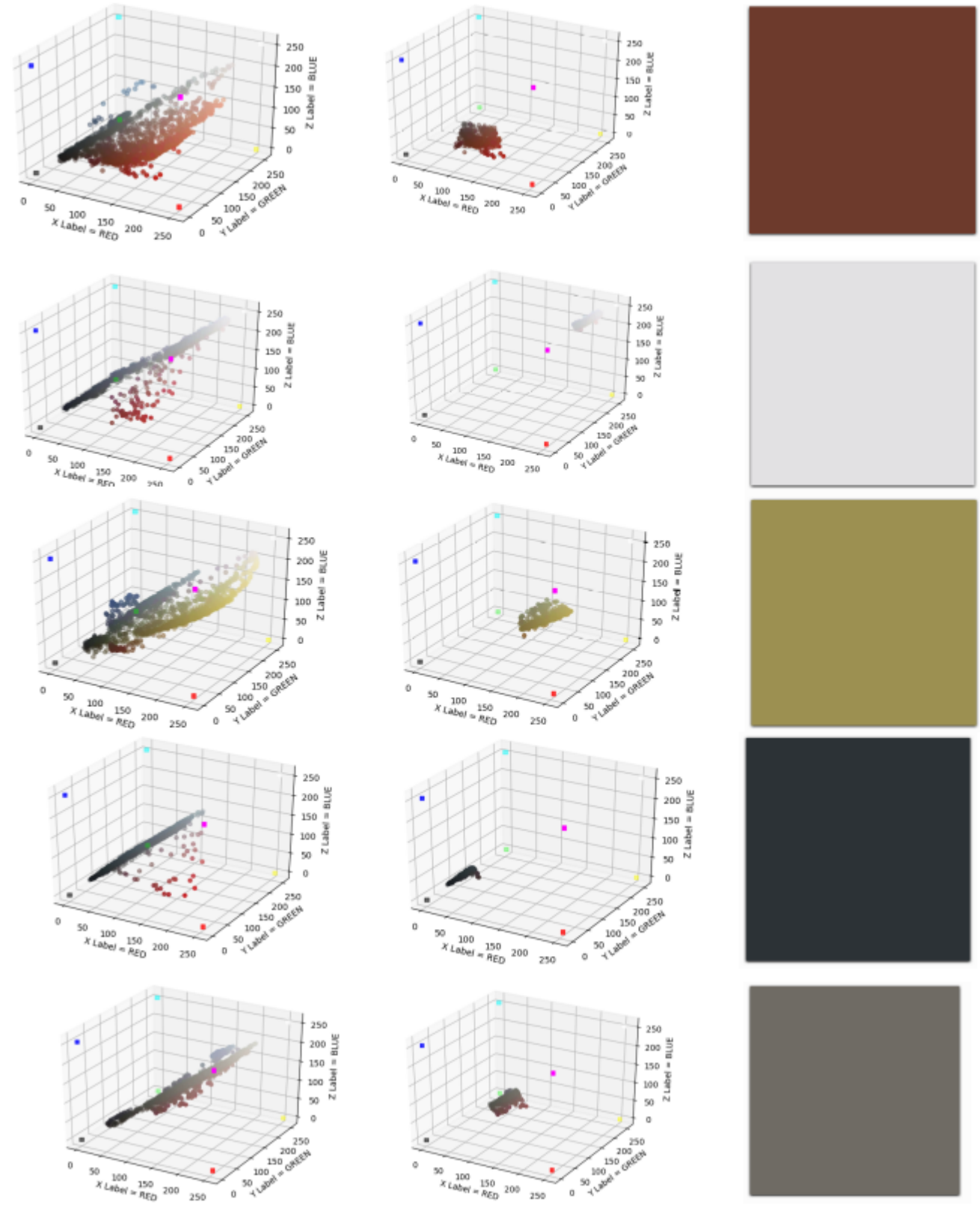

FIGURE 7 – Apprentissage automatique des couleurs

4 Tests et Résultats

Interface

Test 1 :

1. Lancer le logiciel
2. Cliquer sur le bouton *"Sélectionner un véhicule"*
3. Sélectionner une image de la base de test de VeRi
4. Cliquer sur le bouton *"Lancer la réidentification"*
5. Sélectionner dans la liste le véhicule de la liste triée à visualiser

Descripteurs sémantiques : GoogleNet

Les tests pour le réseau de neurones sont directement liées à son entraînement. Si le classifieur à un grand taux de succès sur un grand nombre d'échantillons et de classes cela implique qu'il arrive à identifier les parties importantes des véhicules. Pour maximiser les chances de ré-identification on utilise lors de l'entraînement la métrique de précision des 5 meilleurs résultats, une telle métrique permet de réduire l'effet de sur-apprentissage du réseau.

Le réseau est entraîné à plusieurs reprises sur un jeu d'entraînement qui est mélangé à chaque fois, la validation se fait avec une partie aléatoire du jeu d'entraînement.

```
Train on 5100 samples, validate on 900 samples
Epoch 1/1
5100/5100 [==============================] - 189s 37ms/step - loss: 0.2990 - acc: 0.9716 - top_k_categorical_accuracy: 0.9965 - val_loss: 0.1719 - val_acc: 0.9811 - val_top_k_categorical_accuracy: 0.9978
Train on 5100 samples, validate on 900 samples
Epoch 1/1
5100/5100 [==============================] - 168s 33ms/step - loss: 0.2864 - acc: 0.9739 - top_k_categorical_accuracy: 0.9963 - val_loss: 0.1645 - val_acc: 0.9833 - val_top_k_categorical_accuracy: 0.9978
Train on 5100 samples, validate on 900 samples
Epoch 1/1
5100/5100 [==============================] - 166s 33ms/step - loss: 0.2687 - acc: 0.9759 - top_k_categorical_accuracy: 0.9967 - val_loss: 0.1848 - val_acc: 0.9800 - val_top_k_categorical_accuracy: 0.9989
Train on 5100 samples, validate on 900 samples
Epoch 1/1
5100/5100 [==============================] - 168s 33ms/step - loss: 0.2633 - acc: 0.9759 - top_k_categorical_accuracy: 0.9959 - val_loss: 0.1666 - val_acc: 0.9844 - val_top_k_categorical_accuracy: 0.9989
Train on 5100 samples, validate on 900 samples
Epoch 1/1
5100/5100 [==============================] - 168s 33ms/step - loss: 0.2652 - acc: 0.9753 - top_k_categorical_accuracy: 0.9973 - val_loss: 0.1595 - val_acc: 0.9811 - val_top_k_categorical_accuracy: 0.9989
Train on 5100 samples, validate on 900 samples
Epoch 1/1
5100/5100 [==============================] - 168s 33ms/step - loss: 0.2458 - acc: 0.9778 - top_k_categorical_accuracy: 0.9988 - val_loss: 0.1421 - val_acc: 0.9911 - val_top_k_categorical_accuracy: 1.0000
Train on 1484 samples, validate on 262 samples
Epoch 1/1
1484/1484 [==============================] - 49s 33ms/step - loss: 0.2496 - acc: 0.9825 - top_k_categorical_accuracy: 0.9980 - val_loss: 0.1357 - val_acc: 0.9885 - val_top_k_categorical_accuracy: 1.0000
Epoch 0 effectuer
```

FIGURE 8 – Résultat pour la dernière epoch.
Les informations importantes sont val_acc et val_top_k_categorical_accuracy qui correspondent au taux de classification réussi avec succès sur le jeu de validation. Ici avec un top 5 on est presque assuré de tomber sur la bonne classe.

Descripteurs de textures : BOW-SIFT

Le module **testsDescripteursSift.py** permet de tester toutes les fonctions présentées dans la section 3. Les tests présentés en suivant on été réalisé sur les 5 premières images du corpus d'entraînement, afin de valider les fonctions sans avoir des temps de calcul trop important. Pour mesurer le temps d'exécution de chaque fonction nous utilisons la librairie **time**.

Premier scénario de test :
Pour ce premier scénario de test le modèle k-moyenne n'est pas enregistré dans un fichier puis rechargé par la suite, car le module pickle n'a pas pu être installé sur la machine ayant servit à tester ce scénario.

1. indiquer les chemins des images d'entrainement, des fichiers contenant les descripteurs sift et des fichier contenant les descripteurs BOWSIFT.
2. initialiser le nombre d'image à tester à 5, soit indDIm = 0, indFIm=5, pas =1.
3. k = 1000 : Pour calculer le modèle d'apprentissage par k-moyenne, le nombre de classes k est contraint en fonction du nombre d'éléments dans le dictionnaire.
4. lister les noms des fichier image avec la fonction listerContenuFichier(cheminFichier) du module traitementFichiers.py
5. Créer les fichier des descripteurs sifts des 5 images.
6. extraire le dictionnaire de ces fichiers
7. calculer le modèle d'apprentissage non supervisé
8. créer les fichiers contenant les descripteurs BOW-SIFT des images d'entrainements
9. extraire la liste des descripteurs BOW-SIFT depuis les fichiers.
10. calculer le descripteur BOW-SIFT d'une image de test
11. calculer la distance euclidienne entre le descripteur de l'image de tests et ceux de la base d'entrainement.
12. afficher le tableau des scores.

Résultats du scénario 1 : Le test mené sur 5 images nous montre que chacune des fonctions du module SIFT ont des résultats cohérents.
Pour les 5 images du jeux de tests on a 1507 descripteurs sift au total, et on teste la construction du modèle pour k = 1000. On constate que le temps de calcul

du modèle est d'environ 17 secondes, c'est ce qui prend le plus de temps pour ce module.

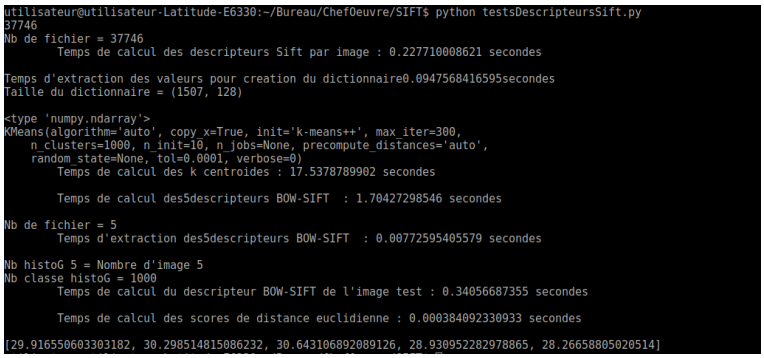

FIGURE 9 – Résultats du test 1

Remarques :

- Dans notre application, nous avons 37746 images. Pour des raisons à la fois matériel et de temps de calcul nous avons opté pour enregistrer le modèle au format *.pkl à l'aide de la librairie pickel afin de ne pas effectuer ces calculs en temps réel dans l'application.
- Lors du lancement du calcul des descripteurs sift des 37746 images redirigé vers des fichiers le temps d'exécution a été de 54 minutes.
- Les différentes tentatives de calcul du modèle sur l'intégralité des descripteurs sift sur machine personnel, et sur google colaboratory en simulant un GPU n'ont pas permis de venir à bout du calcul car trop long.
- Tentatives de calcul du modèle sur 1/4 des images d'entraînements (950 images) : nous avions plus de 3 millions de descripteurs et le calcul s'est arrêté après presque 3h en affichant l'erreur suivante : Error Memory. Cette erreur est du à une saturation de la RAM de la machine utilisée. Ainsi pour résoudre ce problème on privilégie la fonction fit de **MiniBatchKmeans(n_clusters=k,batch_size=bs,verbose=1)** à celle de **Kmeans(n_clusters=k)** de la librairie *sklearn*. On arrive alors à calculer le modèle en 57 minutes environ. Le désavantage de cette méthode est une perte possible de précision pour le calcul des centres.

Résultat du scénario 1 avec MiniBatchKmeans

On remarque que le temps de calcul du modèle pour 5 images, et 1507 descripteurs est de 7 secondes environs, ce qui est 10 secondes de moins que pour la version précédente 4

```
Init 1/3 with method: k-means++
/home/utilisateur/.local/lib/python2.7/site-packages/sklearn/cluster/k_means_.py:1562:
0. Setting it to 3*k
  init_size=init_size)
Inertia for init 1/3: 0.000000
Init 2/3 with method: k-means++
Inertia for init 2/3: 0.000000
Init 3/3 with method: k-means++
Inertia for init 3/3: 0.000000
Minibatch iteration 1/56500: mean batch inertia: 0.000000, ewa inertia: 0.000000
Minibatch iteration 2/56500: mean batch inertia: 28166.666667, ewa inertia: 99.705015
Minibatch iteration 3/56500: mean batch inertia: 24604.333333, ewa inertia: 186.447062
Minibatch iteration 4/56500: mean batch inertia: 15988.000000, ewa inertia: 242.381763
Minibatch iteration 5/56500: mean batch inertia: 30005.000000, ewa inertia: 347.736163
Minibatch iteration 6/56500: mean batch inertia: 54414.666667, ewa inertia: 539.123528
Minibatch iteration 7/56500: mean batch inertia: 0.000000, ewa inertia: 537.215126
Minibatch iteration 8/56500: mean batch inertia: 55708.000000, ewa inertia: 732.509940
Minibatch iteration 9/56500: mean batch inertia: 36713.333333, ewa inertia: 859.875686
[MiniBatchKMeans] Reassigning 1 cluster centers.
Minibatch iteration 10/56500: mean batch inertia: 0.000000, ewa inertia: 856.831879
Minibatch iteration 11/56500: mean batch inertia: 0.000000, ewa inertia: 853.798845
Converged (lack of improvement in inertia) at iteration 11/56500
Computing label assignment and total inertia
MiniBatchKMeans(batch_size=3, compute_labels=True, init='k-means++',
        init_size=None, max_iter=100, max_no_improvement=10,
        n_clusters=1000, n_init=3, random_state=None,
        reassignment_ratio=0.01, tol=0.0, verbose=1)
        Temps de calcul des k centroides : 7.44953799248 secondes

        Temps de calcul des5descripteurs BOW-SIFT  : 2.25079989433 secondes
```

FIGURE 10 – Résultats du test 2

```
Nb de fichier = 5
        Temps d'extraction des5descripteurs BOW-SIFT  : 0.00843000411987 secondes

Nb histoG 5 = Nombre d'image 5
Nb classe histoG = 1000
        Temps de calcul du descripteur BOW-SIFT de l'image test : 0.386022806168 secondes

        Temps de calcul des scores de distance euclidienne : 0.000407218933105 secondes

[0.11973399458202215, 0.1317244983450273, 0.1322268642828423, 0.11672559973034938, 0.12062080901123014]
```

FIGURE 11 – Résultats du test 1

Construction du modèle avec 1/4 de la base d'apprentissage

1. indiquer les chemins des images d'entrainement, des fichiers contenant les descripteurs sift et des fichiers contenant les descripteurs BOWSIFT.
2. initialiser le nombre d'image à tester au nombre image de la base d'apprentissage (37746), soit indDIm = 0, indFIm=37746, pas = 4.
3. k = 10000
4. lister les noms des fichiers image avec la fonction listerContenuFichier(cheminFichier) du module traitementFichiers.py
5. extraire le dictionnaire des fichiers précalculés
6. calculer le modèle d'apprentissage non supervisé

Résultats :
Nous avons ici pour l'extraction des 3223764 descripteurs environ 47 minutes d'exécution, puis 57 minutes pour le calcul du modèle à 10000 classes enregistré au format *.pkl pour pouvoir être facilement récupérable dans l'application pour ne pas calculer le modèle en temps réel (trop long).

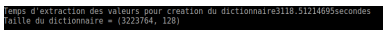

FIGURE 12 – Résultats du test 3

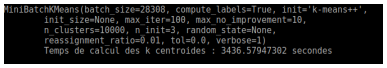

FIGURE 13 – Résultats du test 3

Pré-calcul des descripteurs BOW-SIFT des images de la base d'entraînement :

1. indiquer le chemin du répertoire contenant les fichiers des descripteurs Sift des images de la base d'apprentissage
2. calculer les histogrammes à partir des points d'intérêts de chaque image pour obtenir leur descripteur BOW-SIFT
3. ecrire chaque descripteur dans un fichier *desBOWSift_nomImage*

Résultats : Le pré-calcul des descripteurs BOW-SIFT est primordial, car pour la base d'entrainement qui contient 37746 images dont on souhaite avoir les descripteurs le temps d'exécution a été de 23heures et 26 minutes environ.

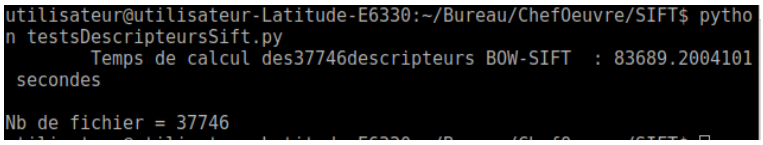

FIGURE 14 – Résultats du test 4

Descripteurs de couleurs : BOW-CN

Le carré pixélisé de la figure 15 montre pour chaque pixel la couleur du véhicule détecté correspondant à une image. Nous pouvons définir une couleur de référence pour le vert qui correspond à la peinture de la voiture afin de mesurer l'erreur. Si on considère que le vert à une référence de RGB=(74,145,63) alors, l'erreur quadratique moyenne de la couleur est de 14.53, contre 27.16 avec le jaune par exemple.

La même démarche est réalisée avec les couleurs suivante : vert, bleu marine,

FIGURE 15 – Exemple de détection avec de véhicule vert

orange,rouge, blanc, jaune, noire. La figure 4 nous montre les résultats. Les véhicules de couleurs foncés sont beaucoup plus sensible aux reflets et donc la détection de la couleur est difficile. La différenciation entre véhicule bleu marine et noir est la plus compliquer puisque le reflet du ciel apport une composante bleu qui une fois moyenné, fait tirer le noir vers le bleu. Les points gris sont la cause de deux choses ; Soit à un mauvais centrage du véhicule dans l'image, et donc, il y a une prédominance de la couleur de la route. Soit beaucoup de reflet blanc dans les vitres. Soit les deux.

FIGURE 16 – Visualisation de la détection de couleur sur un ensemble réduit de 36 véhicules

5 Conclusion

Au départ le projet devait contenir une partie de détection des véhicules dans des vidéos de surveillance. Cette étape aurait permis de constituer notre propre base d'apprentissage, à partir des vidéos de surveillances utilisés dans l'article *Large-scale vehicul re-identification in urban surveillance videos* sous réserve d'une autorisation d'utilisation, à l'aide de Yolo3 spécialisé pour extraire les boites englobantes de véhicules. Cependant cette partie n'a pas pu être réalisé. Elle aurait permis d'améliorer notre prototype, en le rendant plus complet quant à la problématique proposée.

La partie de ré-identification fonctionne cependant de nombreuses améliorations sont à prévoir afin d'obtenir un logiciel utilisable.
Le problème majeur du logiciel réside en ses temps de calcul énorme, même en effectuant le maximum de pré-calcul. La problématique de la ré-identification de véhicule ne permet pas d'avoir des logiciels s'exécutant en temps réel. Néanmoins il n'est pas envisageable d'utiliser des logiciels mettant des jours à effectuer les calculs, surtout en situation de crise. Ainsi une parallélisation lors du calcul des scores des descripteurs sémantique, de couleur et de texture , ainsi que du score Fact fusionnant les trois derniers score pour l'intégralité des images de la base d'apprentissage serait à prévoir.

Afin de limiter au maximum les temps de calcul les différents modèles d'apprentissage utilisés sont pré-calculés. Cependant pour des raisons matériels le calcul des différents modèle n'a pas pu être optimal. Par exemple pour BOW-SIFT le calcul du modèle a pris environ 1h en utilisant une fonction un peu moins précise et seulement le quart des images en raison du très grand nombre de descripteurs sift constituant le dictionnaire d'apprentissage. De plus le calcul des descripteurs BOW-SIFT pour l'intégralité de la base d'apprentissage (ici 37746 images), a duré quasiment 24H, il est important de noter que sur un logiciel utilisable les images comparées seront celles de issues de vidéos surveillances, ce qui constitue des nombres colossaux d'images, et il n'est pas envisageable de fournir un logiciel qui mettent plusieurs jours à être prêt à la recherche de véhicule à chaque nouvelle utilisation.

En conclusion ce prototype mérite de très nombreuses améliorations, cependant en raison des moyens matériels et du temps à notre disposition, les résultats sont satisfaisants.

Bibliographie

[1] Xinchen Liu, Wu Liu, Huadong Ma, Huiyuan Fu. *Large-scale vehicle re-identification in urban surveillance videos*. ICME, 1–6, 2016.

[2] Joseph Redmon, Ali Farhadi. *YOLOv3 : An Incremental Improvement*. arXiv/CoRR, 2018.

[3] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, and Alexander C. Berg. *SSD : Single Shot MultiBox Detector*. ECCV, 21–37, 2016.

[4] Linjie Yang, Ping Luo, Chen Change Loy, Xiaoou Tang. *A Large-Scale Car Dataset for Fine-Grained Categorization and Verification*. arXiv/CoRR, 2015.

[5] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, Andrew Rabinovich. *Going Deeper with Convolutions*.
`https://arxiv.org/abs/1409.4842`