Rapport SY32 P19 : Détection des visages

Haojie LU

I. INTRODUCTION

L'objectif de ce projet est de construire un classifieur d'image par fenêtre glissante capable de détecter les visages dans les photos, en utilisant les méthodes d'extraction des caractéristiques des images et les méthodes d'apprentissage.

Nous avons utilisé le langage Python et ses divers libraires par exemple scikit-learn, scikit-imag, etc. Les fichiers de code Python contient trois scripts :

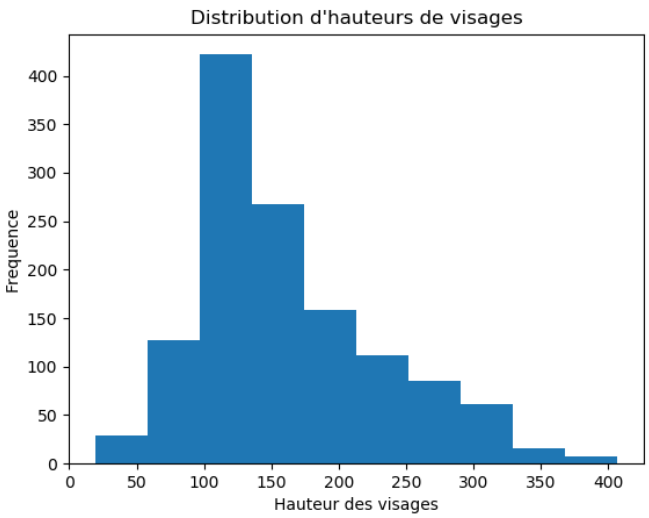
- *train.py* qui doit générer un modèle de classifieur à partir des images d'apprentissage
- *test.py* qui doit générer les détections sur les images de test à partir du modèle générer par *train.py*
- *supplementaire.py* qui contient certains fonctions supplémentaires， par exemple la fonction pour tracer la courbe ROC, etc.

Ce rapport va en détail expliquer comment ces deux fichiers marchent et pourquoi choisissent ces procédés utilisés.

II. PREPARATIONS

Dans cette partie, nous avons un dossier *train* contenant les images d'apprentissage et un fichier *label.txt* contenant les annotations des visages.

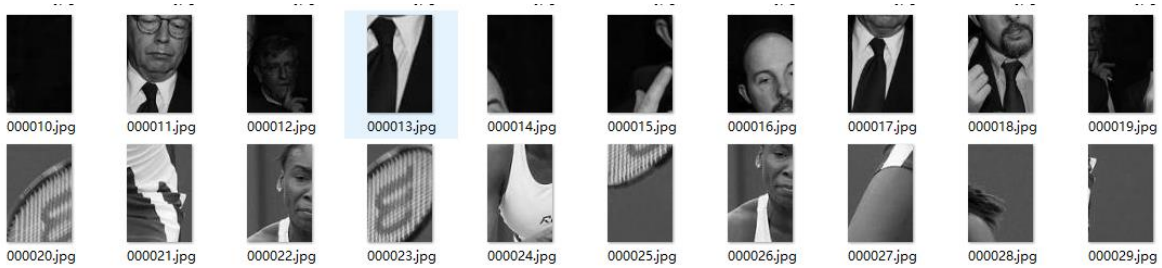
Par calcul, le ratio entre la hauteur et la largeur d'une boîte englobante de visages est approximativement 1.5. Et puis nous avons fait un histogramme de la distribution des hauteurs des visages pour décider une bonne taille fixe de boîte englobante de visage. Pour obtenir une bonne qualité des images, nous avons choisi *90px*60px* comme notre boîte.



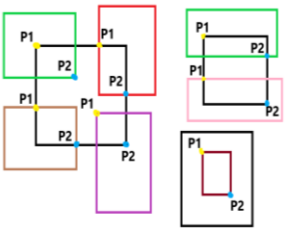
Exemples positifs : Nous avons d'abord découpé les boîtes englobantes de visages afin d'un ratio 1.5 entre leur hauteur et leur largeur. Ensuite, nous extrait tous les visages des images en les redimensionnant à cette taille *90px*60px.* Nous avons eu 1284 visages au total.

Exemples négatifs : Nous avons aléatoirement généré les exemples négatifs ne représentant pas de visage à partir des mêmes images. Nous avons généré dix boîtes négatives par image dont leurs tailles dépendent de celles de boîtes positives dans cette image. Par exemple, s'il y a trois boîtes positives dans une image des tailles (60,40), (120,80) et (100,67), nous allons respectivement obtenir 3, 3 et 4 boîtes négatives de ces trois tailles. En fin, nous avons aussi extrait tous les exemples négatifs en redimensionnant à la même taille des exemples positifs.

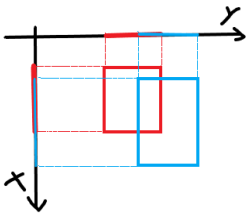
La plus importante partie dans cette procédure est d'éviter le recouvrement parmi les boîtes englobantes. Lors de décider si une boîte négative est conforme ou pas, nous avons tout d'abord calculé le taux d'aire de recouvrement par rapport à chaque boîte positive dans cette image. Nous n'avons que garder la boîte négative dont chaque taux est inférieur à ½. Mais, il y a des images où les visages sont trop grands pour générer 10 boîtes pas de recouvrement et nous donc avons mis une itération maximale 100000. C'est pourquoi que nous n'avons que générer 9920 exemples négatifs.



Pour le calcul du taux d'aire de recouvrement de deux rectangles *(x1, y1, h1, l1) et (x2, y2, h2, l2)*, notre algorithme est composé de :



- Calculer deux points P1 et P2 représentés à côté droit.
  p1_x, p1_y = np.max((x1, x2)), np.max((y1, y2))
  p2_x, p2_y = np.min((x1 + h1, x2 + h2)), np.min((y1 + l1, y2 + l2))
- Juger si l'un rectangle est inclus complètement dans l'autre. Si oui, retourner 1. Sinon continuer.
  (p2_y-p1_y==y1 ***and*** *p2_x-p1_x==x1)* ***or*** *(p2_y-p1_y==y2* ***and*** *p2_x-p1_x==x2)*



- Juger si ces deux rectangles sont intersectés par leurs projections orthogonales dans l'axe x et l'axe y. Sinon, retourner 0. Autrement, calculer *Ajoint* et *AUnion* et retourner *Ajoint/AUnion*.
  min(x1 + h1, x2 + h2) >= max(x1, x2)) ***and*** *(min(y1 + l1, y2 + l2) >= max(y1, y2)*
  *AJoin = (p2_x - p1_x) * (p2_y - p1_y), AUnion = h1 * l1 + l2 * h2 - AJoin*

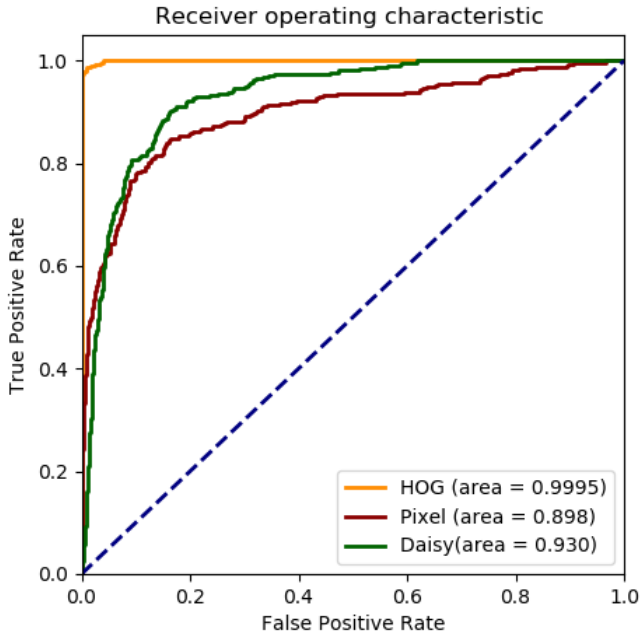
III. DESCRIPTEUR

Cette partie à l'objectif de chercher la meilleure méthode d'extraire les caractéristiques visuelles. Selon les méthodes acquises dans les cours, nous avons les descripteurs Daisy et HOG. La librairie utilisée principalement est *sklearn.feature*.Pour comparer les performances de ces deux descripteurs, nous avons choisi la méthode d'apprentissage SVC avec un noyau linear et le paramètre de pénalité C par défaut. En plus, nous avons aussi entraîné un classifieur en utilisant les pixels directement.

Les paramètres principaux pour *daisy()* et *hog()* sont mis comme ci-dessous :

- *img_daisy=feature.daisy(img,step=10,radius=10,rings=3,histograms=8,orientations=8) , clf = KMeans(n_clusters=20)*
- *feature.hog(img)*

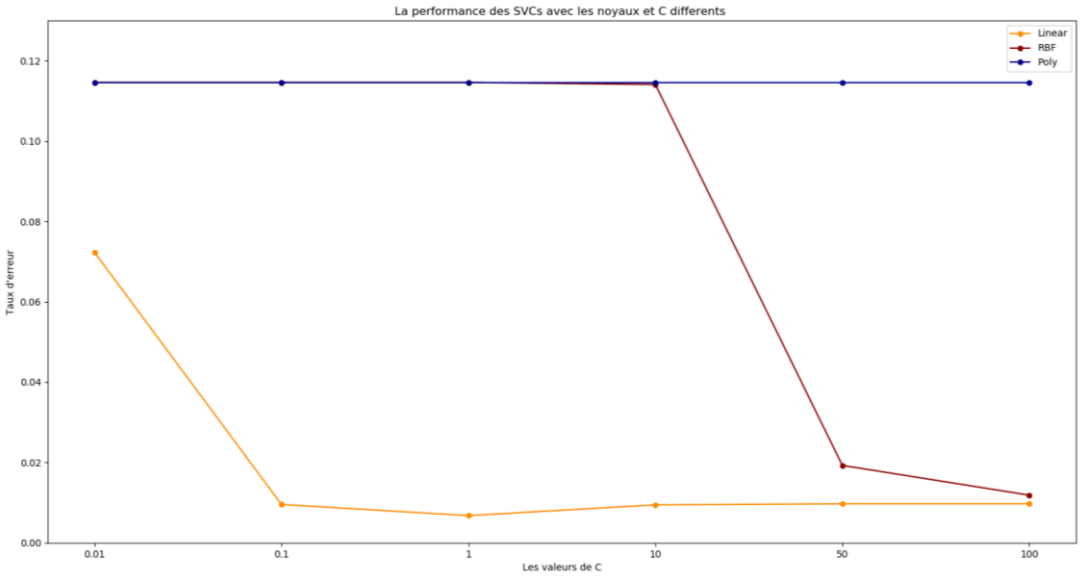
En divisant les données d'apprentissage en deux sous-ensembles : un 75% sous-ensemble d'apprentissage et un 25% sous-ensemble de test, nous avons calculé les valeurs *AUC* et avons tracé les courbes *ROC* montrés en dessous. Les fonctions utilisées sont *auc()*, *roc_curve()* dans la librairie *sklearn.metrics* ainsi que *decision_function()*. Par la courbe, HOG est une très bonne méthode et nous l'avons choisi comme notre descripteur final.



IV. CLASSIFIEUR

Ayant décidé notre descripteur, if faut chercher une méthode d'apprentissage. Ici, nous avons comparé trois méthodes : *SVC*, *Random Forest* et *AdaBoost* and avons calculé leur respective risques par une validation croisée en 5 sous-ensembles. La partition des données est faite par la fonction K*Fold()* de *sklearn.model_selection*.

SVC : C'est très important de choisir les bons hyperparamètres. Nous avons testé 6 valeurs (*0.01,0.1,1,10,50,100*) pour *C* et 3 noyaux (*linear, poly et rbf*) et avons tracé la ligne brisée en dessous. C'est évident qu'elle a la meilleure performance lors de *C=1* avec un noyau *kernel="linear"*.




```
Le taux d'erreur optimal pour SVC : 0.0068
```

AdaBoost

```
Le taux d'erreur pour AdaBoost : 0.0114
```

RandomForest

```
Le taux d'erreur pour RandomForest : 0.0301
```

Ainsi, nous avons choisi HOG+SVC comme notre modèle final. Nous avons entraîné notre classifieur en utilisant tous les données et l'avons gardé dans la disque par la fonction *joblib.dump()* afin que le *sciprt test.py* puisse être exécuté sans le script *train.py*.

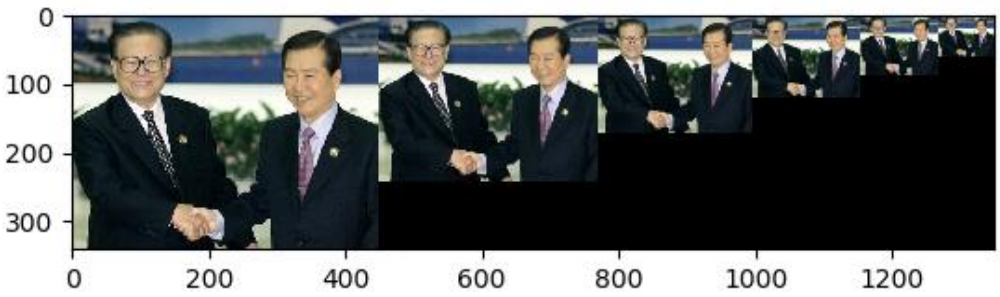
V. DETECTION

Fenêtre glissante

En général, les visages dans les images ont des tailles différentes. On devrait utiliser plusieurs fenêtres glissantes de tailles différentes pour les détecter. Mais il est difficile et moins efficace de redimensionner la fenêtre à cause des variétés de la dimension d'image.

Au lieu de changer la taille de la fenêtre glissante, nous avons choisi de redimensionner l'image d'origine et de garder une fenêtre de taille fixe *90px*60px*. Nous avons utilisé la fonction *pyramid_gaussian() pour réaliser ça.* Ici, nous avons choisi $sqrt(2)$comme l'échelle entre les niveaux et *max_layer=5.* Autrement dire, nous avons également eu 6 fenêtres glissantes différentes : (90,60), (127,84), (180,120), (254,169), (360,240) et (509, 339).

Les fenêtres glissantes peuvent extraire des petites pièces d'images en taille fixe. Généralement, la dernière n'est pas utilisée car la fenêtre glissante est plus grande que l'image après le redimensionnement.



Taille du pas

Dans cette partie, une autre étape importante est du choix de la taille du pas de fenêtre glissante. S'il est trop petit, ça prendra trop de temps pour extraire toutes les pièces. Contrairement s'il est trop grand, on perdra des informations importantes dans les images. La perte d'information va causer une erreur de la détection de position précise du visage.

Tout d'abord, nous avons utilisé *1px* comme pas. Il nous a fait environ 20 minutes de traiter une image. Donc nous avons décidé accroît la taille du pas. Pour réaliser une balance entre l'efficacité et la précision, nous avons décidé d'utiliser deux tailles du pas dynamique pour la colonne et la ligne. « *wstep* » et « *hstep* » sont respectivement le pas horizontal et le pas vertical, 1 par défaut. Ils vont se changer selon la taille des images. Plus petite est l'image redimensionnée, plus petite est la taille du pas. Les formulaires de calcul de la taille du pas sont montrées en dessous :

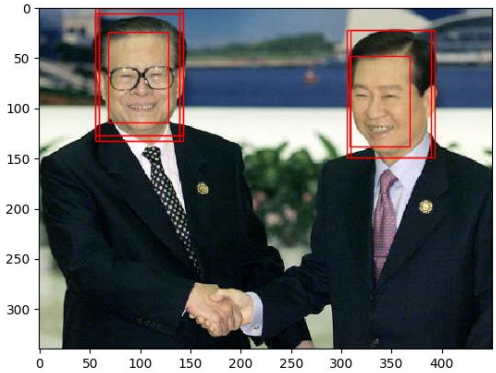
```
wstep= int(round(max(wstep_base / pow(SCALE, i), 1)))
hstep=int(round(max(hstep_base / pow(SCALE, i), 1)))
```

La variable *i* indique le niveau on se trouve. Le « *hstep_base* » et le « *wstep_base* » sont les tailles du pas au début sur l'image d'origine. Nous avons mis 6 ou 4 et 4 ou 3.

Seuil du score

Une fois toutes les petites pièces d'une image sont générées, nous allons les traiter avec l'histogramme de gradient orienté (HOG). Ensuite, nous allons faire les prédictions par *clf.predict()* et obtenir leur scores par la fonction *decision_function()* en utilisant notre classifieur.

Pour le classifieur SVM, toutes les images ayant les scores supérieurs à zéro sont classées dans les visages. Mais nous avons trouvé qu'il y avait plein d'images pas de visage inclus. Par conséquence, nous avons élevé le seuil de score jusqu'à *2.1*. Et puis nous allons calculer et garder les coordonnées du coin supérieur gauche et la taille de chaque boîte dans l'image d'origine ainsi que son score. Avec *matplotlib*, nous avons un exemple 'image ci-dessous :



Jusqu'à cette étape, nous pouvons obtenir plusieurs boîtes englobantes pour un seul visage. Donc, il faut faire une suppression des non-maxima.

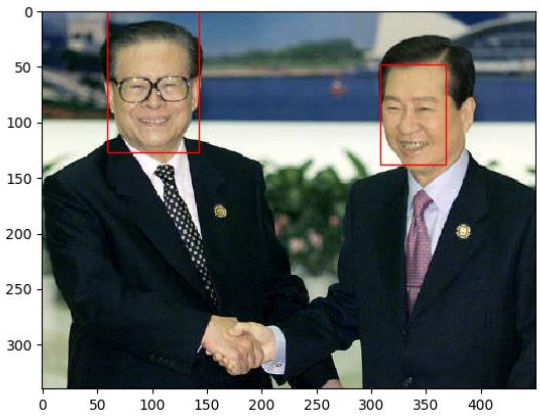
Suppression des non-maxima

Le but de cette partie est de sélectionner la boîte avec le plus élevé score pour chaque paire de boîtes ayant un recouvrement suffisant. Nous avons utilisé la méthode précédente pour calculer le taux de recouvrement. Dans notre algorithme, la procédure de la suppression est composée de 2 étapes.

- La première étape est de supprimer les boîtes avec plus faible score dans la même couche. Dans ce cas, la taille de chaque boîte est la même et nous avons utilisé 50% comme le seuil de détection de recouvrement. Après, nous pouvons obtenir la boîte avec le plus élevé score de chaque visage dans chaque couche.
- La deuxième partie est de sélectionner la meilleure boîte de chaque visage. Mais si le seuil de recouvrement reste toujours *0.5* sans considérant la différence des tailles des boîtes, nous ne pouvons pas supprimer les petites boîtes ayants le taux de recouvrement très bas avec les grandes boîtes, mais qui sont les parties de grandes boîtes. Donc nous avons choisi de mettre un seuil dynamique dépendant du ratio entre les hauteurs de deux boites

$$C = \frac{0.5}{(h_1/h_2)^2}$$ *Où « C » est le seuil,* $h_1 > h_2$.

Par exemple, le seuil de recouvrement pour deux boîtes (90,60) et (180,120) est *0.125*. Finalement, la boîte avec le meilleur score de chaque visage sera gardé. Le résultat final de cet exemple est montré en dessous :



Exécution en parallèle

Le temps de la détection des visages de 500 images est long. Pour réduire le temps d'exécution, nous avons aussi utilisé des scripts pour les exécuter dans plusieurs processus en même temps dans un serveur à distance. Nous avons deux méthodes pour exécuter le script *test.py*.

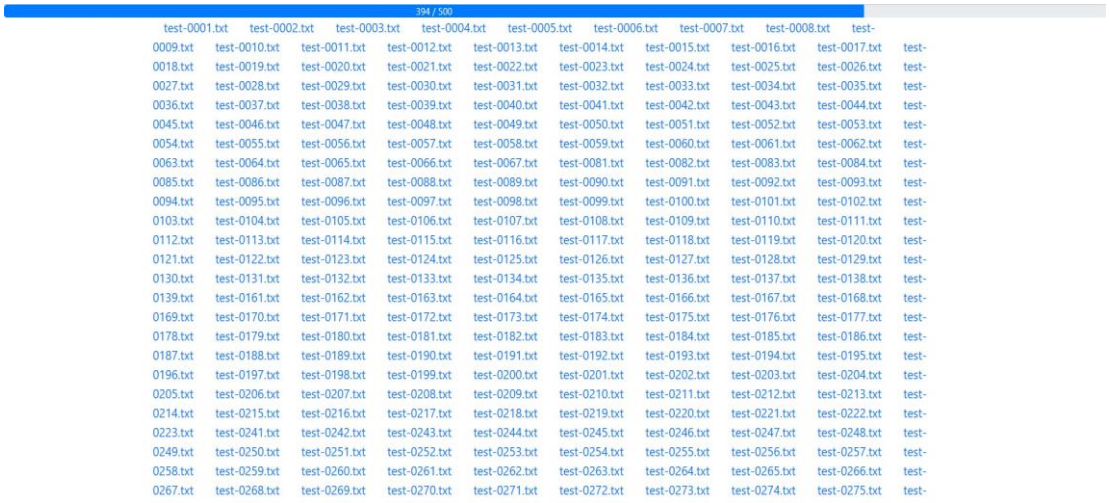
- Pour exécuter séquentiellement les détections dans un seul processus, les résultats sont stockés dans *décision.txt*

  > python test.py

- Pour exécuter une détection d'une seule image, les résultats sont stockés séparément dans les fichiers *test-<indice_d_image>.txt,* il faudra faire un fusionnement manuellement

> python test.py <indice_d_image>

Pour suivre l'avancement, on a également fait une interface Web où se trouve les fichiers sortis. Les résultats plus récents sont mis en http://159.89.99.170:8085/.



VI. Conclusion

Dans ce projet, nous avons bien réalisé la détection des visages en comparant les modèles différents. Nous avons beaucoup amélioré notre performance par ajuster les paramètres, par la taille du pas, seuil du score,etc.

Dans la partie de détection, nous avons rangé des codes et avons donné une méthode d'exécution en parallèle pour réduire le temps.

Tout d'abord, nous avons utilisé 1 pixel comme un pas de fenêtre glissante. L'échec d'exécution a lieu souvent faute de mémoire. Par exemple, dans la première tentative, 22 sur 500 a échoué. Nous avons enregistré les indices d'image 'échec et les avons stockés dans *lack.txt* , ensuite nous avons utilisé la commande pour les réexécuter.

for i in $(cat lack.txt); do python test.py $i; done

Malgré une bonne performance que nous avons eue, il y a encore quelques points restants à améliorer.

- Dans la pyramide des images, nous n'avons diminué que la taille d'images. La performance de détection a une relativement mauvaise pour les visages dont la taille est inférieure à *90px*60px*. Nous pouvons également essayer plusieurs autres échelles, qui peut-être donner une meilleure performance.
- Nous n'avons comparé que 2 descripteurs et 3 méthodes d'apprentissage. Nous pouvons essayer plus de méthodes, par exemple l'apprentissage profond.